ABSTRACT

WILLERT, JEFFREY ALAN. Hybrid Deterministic/Monte Carlo Methods for Solving the Neutron Transport Equation and k-Eigenvalue Problem. (Under the direction of C.T. Kelley.)

The goal of this thesis is to build hybrid deterministic/Monte Carlo algorithms for solving the neutron transport equation and associated k-eigenvalue problem. We begin by introducing and deriving the transport equation before discussing a series of deterministic methods for solving the transport equation. To begin we consider moment-based acceleration techniques for both the one and two-dimensional fixed source problems. Once this machinery has been developed, we will apply similar techniques for computing the dominant eigenvalue of the neutron transport equation. We'll motivate the development of hybrid methods by describing the deficiencies of deterministic methods before describing Monte Carlo methods and their advantages. We conclude the thesis with a chapter describing the detailed implementation of hybrid methods for both the fixed-source and k-eigenvalue problem in both one and two space dimensions. We'll use a series of test problems to demonstrate the effectiveness of these algorithms before hinting at some possible areas of future work.

Hybrid Deterministic/Monte Carlo Methods for Solving the Neutron Transport Equation and k-Eigenvalue Problem

by
Jeffrey Alan Willert

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2013

APPROVED BY:

Yousry Azmy	Mark Hoefer
Dana Knoll	C.T. Kelley Chair of Advisory Committee

DEDICATION

To my parents, for they provided my first experiences with numbers and have supported me every step along this journey.

BIOGRAPHY

Jeffrey Alan Willert was born on September 27, 1986 near Cleveland, Ohio to parents Carol and Kenneth Willert, along side younger siblings Katherine and Matthew. He spent his entire childhood growing up in Chagrin Falls, Ohio and attended the Kenston Local School District from kindergarten through high school. Upon graduation from high school in 2005, Jeff attended The College of Wooster. He graduated from The College of Wooster in 2009 earning a bachelors degree in Mathematics. Later in 2009 he moved to Raleigh, North Carolina to attend North Carolina State University to pursue a doctorate in Applied Mathematics. Throughout his time in graduate school, Jeffrey regularly moved back and forth between Raleigh and Los Alamos, New Mexico to spend time working at Los Alamos National Laboratory.

Jeff's love for mathematics has existed since early childhood. Jeff can remember a car ride during his early years when he recited the geometric series "1, 2, 4, 8, 16, ..." well into the thousands. While probably slightly concerned about their "different" child, Jeff's parents provided his first introduction to mathematics and have been supportive throughout his entire educational process. Jeff's love and talent for mathematics has no doubt been fostered my several wonderful teachers along the way, most notably Mr. Koltas, who was the first teacher to challenge Jeff in mathematics and only made his love for the subject grow deeper. Upon leaving high school, there was no doubt that Jeff would pursue a career in mathematics.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Tim Kelley for the wonderful opportunity he provided me with this project and for all of his help and guidance along the way. It has been a true pleasure working with him and I attribute much of my success to the excellent advising which he provided. I would like to thank the rest of my committee for taking the time to help me through this process. The classes which I have taken from Dr. Kelley, Dr. Azmy and Dr. Hoefer have challenged me and, in many ways, provided a foundation for my success as a researcher. While Dr. Knoll has never stood at the front of one of my classes, the education he has provided me should not be understated - much of what I've learned in this field can attributed directly to my interactions with him.

Additionally, I must thank everyone at Los Alamos National Laboratory who helped me along the way. A special thanks must be delivered to Dr. Dana Knoll for acting much like a second research advisor and taking his time to guide my research and providing me all of the opportunities at the lab. Thank you to Rysouke Park for his neutronics expertise, his help working with Trilinos, and for countless scientific discussions which undoubtedly helped my research progress. Finally, thank you to William Taitano, Joshua Payne and Christopher Newman for additional Trilinos help and high-performance computing guidance.

I would also like to take a moment to acknowledge the entire mathematics department at The College of Wooster. Thank you for helping me put myself in a position to succeed in mathematics graduate school. Thank you especially to Dr. Ramsay, Dr. Pasteur and Dr. Pierce, each of whom played an important role getting me to where I am today. I must thank Dr. Ramsay for the discussions which lead me to attend The College of Wooster and all of the effort he puts in each year to making the AMRE program a success. Thank you to Dr. Pasteur for my first introduction into applied mathematics and for suggesting I consider North Carolina State University for graduate school. Lastly, a most special thank you to Dr. Pierce - she guided me from my very first mathematics class at The College of Wooster through my senior Independent Study Thesis. Dr. Pierce knew I was going to be successful in graduate school before I even considered attending.

Lastly, I'd like to thank all of my friends and family who have helped me get to where I am today. A special thanks to my parents who have always offered their continuous support in an abundance of ways. Thank you for fostering my love for math - some of my earliest memories involve sitting in a car babbling math or being given large division problems to keep me entertained. An additional thanks for making my education at The College of Wooster possible. A thanks to my great friends, Bob and Hallie, who still hang out with me despite me having made them listen my ramblings about research over the years. Finally, thank you

to all of my peers and professors at North Carolina State University. A special thanks to all of my friends who have helped me clear my mind outside of the office, especially through our wonderful departmental intramural teams. Additionally, thank you to Lake Bookman and Lucas Van Blaircum for the weekly mexican lunches, which undoubtedly included some of the most intelligent mathematics discussions which I've had over the past few years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The goal of our work is to construct, or improve, methods for solving the neutron transport equation in the most efficient and precise manner possible. This equation governs the distribution of neutrons in space, angle and energy and its solution is essential for constructing and calibrating nuclear reactors, radiation shielding, medical imaging, and a wide range of other applications. Generally, we let $\psi(\vec{r}, E, \hat{\Omega}, t)\ dV\ d\hat{\Omega}\ dE\ dt$ denote the total path length traversed by all particles within some differential volume dV about the position $\vec{r}$ in $\mathbb{R}^3$, with energies within dE about E, traveling in directions $d\hat{\Omega}$ about $\hat{\Omega}$ during a time dt about t. In the most general case, the neutron transport equation is

$$\begin{aligned}\frac{1}{v}\frac{\partial\psi}{\partial t} + \hat{\Omega}\cdot\nabla\psi + \Sigma_t(\vec{r}, E)\psi(\vec{r}, E, \hat{\Omega}, t) \\ = \int_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \Sigma_s(E' \to E, \hat{\Omega}' \to \hat{\Omega})\psi(\vec{r}, E', \hat{\Omega}', t) + \\ \frac{\chi(\vec{r}, E, t)}{4\pi}\int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \nu(\vec{r}, E', t)\Sigma_f(\vec{r}, E', t')\psi(\vec{r}, E', \hat{\Omega}', t) + s(\vec{r}, E, \hat{\Omega}, t), \quad (1.1)\end{aligned}$$

where $\vec{r} = (x, y, z)^T$ describes the position within the medium. $\hat{\Omega}$ describes the direction that particles are traveling. We represent integration over all directions in the following manner:

$$\int_{4\pi} \cdot\ d\hat{\Omega} \equiv \int_0^{2\pi}\int_0^{\pi} \cdot\ \sin(\theta) d\theta d\omega$$

The function s is a known source. We also must specify a boundary condition

$$\psi(\vec{r}_s, E, \hat{\Omega}, t) = \psi_b(\vec{r}_s, E, \hat{\Omega}, t) \quad (1.2)$$

for $\hat{\Omega}\cdot\hat{e}_s < 0$ for all $\vec{r}_s$ on the surface in the case of a non-reentrant boundary where $\hat{e}_s$ is the unit outward normal. Throughout this thesis, we will assume all boundaries are non-reentrant.

For the time dependent form, we must also specify an initial condition:

$$\psi(\vec{r}, E, \hat{\Omega}, 0) = \psi_0(\vec{r}, E, \hat{\Omega}). \tag{1.3}$$

Before we continue, let's discuss the derivation of the neutron transport equation [7].

1.1 Deriving the Transport Equation

Our goal in this study is to determine the distribution of neutrons in a system. Given some system configuration, we need to be able to accurately predict the balance of neutrons and this can be accomplished by solving the neutron transport equation. However, before we can use this equation, we must understand every piece of it. To aid in our understanding of the equation, we'll go through a step by step derivation. We can describe the distribution of neutrons in space, angle, and energy by a function $n = n(\vec{r}, E, \hat{\Omega}, t)$.

The balance of neutrons in a reactor is governed by the material composition within the reactor as well as the way in which neutrons interact with matter, neutron sources and contributions from external boundaries. It should be clear that neutrons will interact differently with different materials based upon their physical structure. It will be beneficial in our study to begin this derivation by looking at several material properties that, in effect, help us describe the distribution of neutrons in a system.

1.1.1 Basic Definitions

When a neutron undergoes an interaction with matter, several things can happen. First, the neutron can scatter. By this we mean that the neutron hits a nucleus and bounces off with a new direction and energy. Secondly, a neutron can be absorbed by a nucleus. In some absorption reactions, the nucleus can undergo fission and produce more neutrons. For now, we'll simply stick with these three options: scattering, absorption, and fission. We begin with a few definitions:

Definition 1. *Σ_s is called the macroscopic scattering cross-section. Σ_s is equal to the probability of undergoing a scattering reaction per unit path length traveled.*

Definition 2. *Σ_a is called the macroscopic absorption cross-section. Σ_a is equal to the probability of undergoing an absorption reaction per unit path length traveled.*

Definition 3. *Σ_f is called the macroscopic fission cross-section. Σ_f is equal to the probability of undergoing a fission reaction per unit path length traveled. The quantity ν denotes the mean number of neutrons created per fission event.*

Definition 4. *Σ_t is called the macroscopic total cross-section and is equal to $\Sigma_s + \Sigma_a$. Σ_t can be interpreted as the probability per unit path length traveled that a neutron will incur a reaction with a nucleus in the medium.*

Let us discuss these three cross-sections in a bit more detail before moving on. First, it is important to note that Σ_s, Σ_a and Σ_t are all functions of energy, position, and time. In addition, Σ_s is a function of the incoming angle, as well as the post-interaction angle and energy. It makes sense that these quantities should differ for different materials that comprise the reactor. Furthermore, as the energy of a neutron changes, its speed changes and this will affect the probability of undergoing various types of reactions. Lastly, as time progresses, the material properties of the reactor will change. This is immediately clear when we consider the burn-up of fuel or heating of materials. For this reason, we usually represent $\Sigma_t = \Sigma_t(\vec{r}, E, t)$ and follow a similar convention for Σ_s and Σ_a. The scattering cross-section has a more complicated representation and will be discussed in greater detail later.

In order to understand these cross-sections further, let us look at a simple example. Suppose $\Sigma_s/\Sigma_a = 9$. This tells us that on average, in an infinite medium a neutron will undergo 9 scattering interactions before it is absorbed. If $\Sigma_s/\Sigma_a = 9$, we can see that $\Sigma_s/\Sigma_t = .9$. This tells us that given an interaction, the probability that the neutron scatters is .9.

Let us further examine the quantity Σ_a. Σ_a refers to the probability of undergoing an absorption reaction per unit path length. Now, if we let v denote the speed of a particle, we can see that $\Sigma_a v$ is equal to the mean number of absorption reactions per second. If we have n neutrons per cm^3, $\Sigma_a vn$ is equal to the absorption reaction rate density, that is, the rate at which neutrons are being absorbed per cm^3. Then, if V represents some volume in the reactor, $Vv\Sigma_a n$ represents the loss rate in that volume due to absorption.

Let ν be equal to the average number of neutrons created per fission event. Then, the quantity $Vv\Sigma_f \nu n$ is equal to the fission production rate in V, or the rate at which neutrons are produced due to fission. This type of analysis will help us throughout our derivation of the neutron transport equation.

As previously mentioned, $\hat{\Omega}$ refers to the direction that a neutron travels. In general $\hat{\Omega} = \hat{\Omega}(\theta, \omega) = [\sin(\theta)\cos(\omega), \sin(\theta)\sin(\omega), \cos(\theta)]^T$. We notice that $\hat{\Omega}$ is a unit vector in the direction of motion.

During a scattering reaction, it is common for both energy and direction to change. Generally, we let E and $\hat{\Omega}$ represent the initial energy and direction, respectively, and E' and $\hat{\Omega}'$ denote the energy and direction after the collision. For this reason, $\Sigma_s = \Sigma_s(E' \to E, \hat{\Omega}' \to \hat{\Omega})$. However, we can break this scattering cross-section down into more understandable pieces.

In general, we can say that $\Sigma_s(E', \hat{\Omega}' \to E, \hat{\Omega}) = \Sigma_s(E') \cdot P(E', \hat{\Omega}' \to E, \hat{\Omega})$, where $\Sigma_s(E')$ is the probabilily that a neutron with energy E' will scatter when it undergoes an interaction. $P(E', \hat{\Omega}' \to E, \hat{\Omega})$ is the probability that, given a scattering reaction occurs, the particle will

change direction from $\hat{\Omega}'$ to $\hat{\Omega}$ and change energy from E' to E. Using our knowledge of probability, it is easy to see that

$$\int_0^\infty dE \int_{4\pi} d\hat{\Omega} P(E', \hat{\Omega}' \to E, \hat{\Omega}) = 1. \tag{1.4}$$

Now, we can see that

$$\begin{aligned} \int_0^\infty dE \int_{4\pi} d\hat{\Omega} \Sigma_s(E' \to E, \hat{\Omega} \to \hat{\Omega}') &= \int_0^\infty dE \int_{4\pi} d\hat{\Omega} \Sigma_s(E') \cdot P(E', \hat{\Omega}' \to E, \hat{\Omega}) \\ &= \Sigma_s(E') \int_0^\infty dE \int_{4\pi} d\hat{\Omega} P(E', \hat{\Omega}' \to E, \hat{\Omega}) \\ &= \Sigma_s(E') \end{aligned}$$

1.1.2 Derivation

As was previously mentioned, $n(\vec{r}, \hat{\Omega}, E, t)$ is the neutron density function, sometimes referred to as the neutron distribution function. This function n is such that $n(\vec{r}, \hat{\Omega}, E, t)\ d^3r\ d\hat{\Omega}\ dE$ is equal to the expected number of particles in some volume d^3r about $\vec{r}$, with energies in dE about E, traveling in directions $d\hat{\Omega}$ about $\hat{\Omega}$ at time t.

In order to derive the transport equation, we need a balance relation describing the movement of neutrons in phase space. Let us define a volume in phase space, P. Our phase space consists of 3 variables (6 dimensions), $\vec{r}$, $\hat{\Omega}$, and E. We will refer to the quantity $d^3r\ d\hat{\Omega}\ dE$ as the differential volume in phase space.

In its most basic form, the neutron balance equation can be given by:

$$[\text{change rate in } P] = [\text{neutrons entering } P] - [\text{neutrons leaving } P].$$

If we want to determine the number of neutrons in P, we can integrate $n(\vec{r}, \hat{\Omega}, E, t)$ over the spatial volume ΔV, range of directions $\Delta\hat{\Omega}$ and energy range ΔE. Let us define $N(t)$ as the number of neutrons in P at time t.

$$N(t) = \int_{\Delta V} \int_{\Delta E} \int_{\Delta\hat{\Omega}} n(\vec{r}, \hat{\Omega}, E, t) d\hat{\Omega} dE d^3r \tag{1.5}$$

Now, we can represent the change rate in the phase volume P as dN/dt.

Methods in which neutrons can enter the phase volume P:

1. Neutrons can be supplied by an external source
2. Neutrons can enter due to fission

3. Neutrons can scatter into the phase volume (inscattering)

4. Neutrons can leak into the phase volume (neutrons change location in space)

Methods in which neutrons can leave the phase volume P:

1. Neutrons can scatter out of the phase volume (outscattering)

2. Neutrons can be absorbed in the phase volume

3. Neutrons can leak out of the phase volume (neutrons change location in space)

If we can describe the rate at which all of these gains and losses occur, we will have a balance relation describing the distribution of neutrons in phase space. We will derive some of these terms, while others will simply be given. For most terms, the derivations are quite similar. We'll look at each term in the equation in terms of gains and losses.

Gains

1. **External Source:** This requires no formal derivation. This is simply a source term describing the rate at which neutrons enter the phase volume due to an specified external source. When solving the transport equation, this will be considered data like the material properties in the system. This source is described by:

$$S_{ext}(\vec{r}, \hat{\Omega}, E, t) = \frac{\#\text{ of neutrons}}{cm^3 \cdot MeV \cdot ster \cdot sec} \tag{1.6}$$

where MeV stands for mega-electron volts and *ster* stands for steradians, the SI unit of solid angle, the three-dimensional analog to radians which describe the span of an angle in the plane.

Now, the rate at which neutrons enter the phase box due to this external source is given by:

$$\int_{\Delta V} d^3r \int_{\Delta E} dE \int_{\delta\hat{\Omega}} d\hat{\Omega}\ S_{ext}(\vec{r}, \hat{\Omega}, E, t) \tag{1.7}$$

2. **Fission Source:** Recall that Σ_f is equal to the number of interactions per unit path length of a neutron and $v(E)\Sigma_f$ is equal to the interaction rate per neutron (here $v = v(E)$ since the speed is exactly determined by the energy of a neutron as it is assumed to behave like a classical point particle) [21]. Now, the quantity

$$v(E)\Sigma_f(\vec{r}, E, t)n(\vec{r}, \hat{\Omega}, E, t)d^3r d\hat{\Omega} dE \tag{1.8}$$

represents the fission rate in d^3r about $\vec{r}$ caused by neutrons with energies within dE about E with directions $d\hat{\Omega}$ about $\hat{\Omega}$ at time t. Thus,

$$\int_{\Delta V} d^3r \int_{\Delta E} dE \int_{\delta\hat{\Omega}} d\hat{\Omega}\ \ v(E)\Sigma_f(\vec{r},E,t)n(\vec{r},\hat{\Omega},E,t) \tag{1.9}$$

describes the fission rate within the phase box. However, we are not just interested in the fission rate within the phase box, we are interested in the rate at which neutrons are produced due to fission within the phase box.

We can see that the number of neutrons produced due to fission events in d^3r about $\vec{r}$ caused by neutrons with energies in dE about E and with directions $d\hat{\Omega}$ about $\hat{\Omega}$ at time t is given by

$$\nu(\vec{r},E,t)\Sigma_f(\vec{r},E,t)v(E)n(\vec{r},\hat{\Omega},E,t)\ d^3r\ d\hat{\Omega}\ dE \tag{1.10}$$

where, as a reminder to the reader, ν is the mean number of fission neutrons produced per fission event.

This is not enough to determine how many neutrons are entering the phase box, however. We must know more information about the neutrons being produced; we must know the direction they are traveling when they are produced as well as their energy. We can assume that the fission neutron directions are distributed isotropically. That is, there is no preference of direction for fission neutrons, or one could say that there is equal probability of having any initial direction when produced due to fission. However, fission neutrons may be born with a variety of energies, and this spectrum is described by $\chi(E)$. Therefore, $\chi(E)dE$ describes the fraction of fission neutrons born with energies in dE about E. Now, we're able to described the rate at which neutrons enter the phase box due to fission. This quantity is given by:

$$\int_{\Delta V} d^3r \int_{\Delta E} dE \int_{\delta\hat{\Omega}} d\hat{\Omega}\ \ \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \ \nu(\vec{r},E',t)\Sigma_f(\vec{r},E',t')v(E')n(\vec{r},\hat{\Omega}',E',t)$$

It is important to make the distinction in the above quantity, E' describes the energy of the neutron causing the fission event, while E describes the energy of the fission neutrons.

3. **Inscattering:** The term *inscattering* refers to neutrons which scatter into the phase volume. This essentially means that after undergoing a scattering reaction, the neutrons have angles in $\Delta\hat{\Omega}$ about $\hat{\Omega}$ and energies in ΔE about E. Given the previous derivations, this quantity is relatively easy to describe:

$$\int_{\Delta V} d^3r \int_{\Delta E} dE \int_{\Delta\hat{\Omega}} d\hat{\Omega} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \ \Sigma_s(\vec{r},E',\hat{\Omega}'\to E,\hat{\Omega},t)v(E)n(\vec{r},\hat{\Omega}',E',t)$$

The term above describes the rate at which neutrons scatter into the phase box.

4. **Leakage into the Phase Box:** Leakage into and out of the phase box will be handled simultaneously in its own section.

Losses

1. **Absorption:** When neutrons undergo collisions, it is possible that the neutron is absorbed. Recall that the quantity

$$\Sigma_a(\vec{r}, E, t)v(E)n(\vec{r}, \hat{\Omega}, E, t)$$

is the rate at which neutrons are absorbed per cm^3. Therefore, the rate at which neutrons are absorbed within the phase box is given by

$$\int_{\Delta V} d^3r \int_{\Delta E} dE \int_{\Delta\hat{\Omega}} d\hat{\Omega}\ \ \Sigma_a(\vec{r}, E, t)v(E)n(\vec{r}, \hat{\Omega}, E, t).$$

2. **Outscattering:** The term *outscattering* refers to the loss of neutrons which begin inside the phase volume but leave the phase volume due to scattering. This quantity is far more simple than the *inscattering* term because we do not concern ourselves with the final direction and energy of the scattered neutrons. We only care about the rate at which neutrons scatter within the phase volume. While some neutrons may undergo a scatter and remain within the phase volume, this quantity is taken care of in the inscattering term.

   We can describe the rate at which neutrons outscatter by

$$\int_{\Delta V} d^3r \int_{\Delta E} dE \int_{\Delta\hat{\Omega}} d\hat{\Omega}\ \ \Sigma_s(\vec{r}, E, t)v(E)n(\vec{r}, \hat{\Omega}, E, t).$$

Leakage

Instead of computing different quantities as far as gains and losses, we will instead derive one term that incorporates the net leakage of neutrons in our system.

Consider some arbitrary surface dS with unit normal vector $\vec{e_n}$. We can describe the net rate at which neutrons cross the area dS around $\vec{r}$ having energies within dE about E, traveling in directions within $d\hat{\Omega}$ about $\hat{\Omega}$ at time t by

$$\hat{\Omega} \cdot \vec{e_n}\ v(E)n(\vec{r}, E, \hat{\Omega}, t)\ dS\ dE\ d\hat{\Omega}.$$

Now, the rate at which neutrons leave the phase box is given by

$$\int_{S_{\Delta V}} ds \int_{\Delta E} dE \int_{\Delta \hat{\Omega}} d\hat{\Omega}\ \hat{\Omega} \cdot \vec{e_n}\ v(E) n(\vec{r}, E, \hat{\Omega}, t)$$

where $S_{\Delta V}$ denotes the surface of ΔV. If we apply the divergence theorem, we can get this quantity in more recognizable (and useful) terms:

$$\int_{\Delta V} d^3 r \int_{\Delta E} dE \int_{\Delta \hat{\Omega}} d\hat{\Omega}\ \ \hat{\Omega} \cdot \nabla v(E) n(\vec{r}, E, \hat{\Omega}, t)$$

1.1.3 The Transport Equation

Now that we have all of our gains, losses and leakage terms, we can put it all together to form the neutron transport equation. We will write this as:

$$[\text{rate of change}] + [\text{loss rate}] = [\text{gain rate}]. \tag{1.11}$$

We have:

$$\begin{aligned}
&\int_{\Delta V} d^3 r \int_{\Delta E} dE \int_{\Delta \hat{\Omega}} d\hat{\Omega}\ \ \frac{\partial n(\vec{r}, E, \hat{\Omega}, t)}{\partial t} + \hat{\Omega} \cdot \nabla v(E) n(\vec{r}, E, \hat{\Omega}, t) \\
&\qquad + \Sigma_s(\vec{r}, E, t) v(E) n(\vec{r}, \hat{\Omega}, E, t) + \Sigma_a(\vec{r}, E, t) v(E) n(\vec{r}, \hat{\Omega}, E, t) \\
&= \int_{\Delta V} d^3 r \int_{\Delta E} dE \int_{\Delta \hat{\Omega}} d\hat{\Omega} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega}, t) v(E) n(\vec{r}, \hat{\Omega}', E', t) \\
&\qquad + \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \ \nu(\vec{r}, E', t) \Sigma_f(\vec{r}, E', t') v(E') n(\vec{r}, \hat{\Omega}', E', t) \\
&\qquad\qquad + S_{ext}(\vec{r}, \hat{\Omega}, E, t)
\end{aligned}$$

Since ΔV, ΔE and $\Delta \hat{\Omega}$ are all arbitrary we can conclude that the two integrands are equivalent everywhere within our domain. Thus, we have

$$\begin{aligned}
&\frac{\partial n(\vec{r}, E, \hat{\Omega}, t)}{\partial t} + \hat{\Omega} \cdot \nabla v(E) n(\vec{r}, E, \hat{\Omega}, t) \\
&\qquad + \Sigma_s(\vec{r}, E, t) v(E) n(\vec{r}, \hat{\Omega}, E, t) + \Sigma_a(\vec{r}, E, t) v(E) n(\vec{r}, \hat{\Omega}, E, t) \\
&\qquad = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega}, t) v(E) n(\vec{r}, \hat{\Omega}', E', t) \\
&\qquad + \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}'\ \ \nu(\vec{r}, E', t) \Sigma_f(\vec{r}, E', t') v(E') n(\vec{r}, \hat{\Omega}', E', t) \\
&\qquad\qquad + S_{ext}(\vec{r}, \hat{\Omega}, E, t)
\end{aligned}$$

Furthermore, we know that $\Sigma_a + \Sigma_s = \Sigma_t$, we can simplify slightly

$$\begin{aligned}\frac{\partial n(\vec{r}, E, \hat{\Omega}, t)}{\partial t} + \hat{\Omega} \cdot \nabla v(E) n(\vec{r}, E, \hat{\Omega}, t) + \Sigma_t(\vec{r}, E, t) v(E) n(\vec{r}, \hat{\Omega}, E, t) \\ = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega}, t) v(E) n(\vec{r}, \hat{\Omega}', E', t) \\ + \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \nu(\vec{r}, E', t) \Sigma_f(\vec{r}, E', t') v(E') n(\vec{r}, \hat{\Omega}', E', t) + S_{ext}(\vec{r}, \hat{\Omega}, E, t)\end{aligned}$$

Now, we see that most terms include the quantity $v(E)n(\vec{r}, E, \hat{\Omega}, t)$. We generally simplify the transport equation by defining $\psi(\vec{r}, E, \hat{\Omega}, t) = v(E)n(\vec{r}, E, \hat{\Omega}, t)$. ψ is referred to as the angular flux. Substituting this quantity into the transport equation yields the following integro-differential equation:

$$\begin{aligned}\frac{1}{v(E)} \frac{\partial \psi(\vec{r}, E, \hat{\Omega}, t)}{\partial t} + \hat{\Omega} \cdot \nabla \psi(\vec{r}, E, \hat{\Omega}, t) + \Sigma_t(\vec{r}, E, t) \psi(\vec{r}, E, \hat{\Omega}, t) \\ = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega}, t) \psi(\vec{r}, E', \hat{\Omega}', t) \\ + \frac{\chi(\vec{r}, E, t)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \nu(\vec{r}, E', t) \Sigma_f(\vec{r}, E', t') \psi(\vec{r}, E', \hat{\Omega}, t) + S_{ext}(\vec{r}, \hat{\Omega}, E, t) \quad (1.12)\end{aligned}$$

We will refer to Equation 1.12 as the three-dimensional, time-dependent, energy-dependent, angularly-dependent neutron transport equation, or the 3-D neutron transport equation for short.

1.1.4 Initial and Boundary Conditions

It is also important to develop boundary conditions for the neutron transport equation. We'll focus primarily on three types of boundary conditions [21]:

1. Known surface source
2. Vacuum boundary conditions (no incoming neutrons)
3. Reflective boundary conditions

We'll discuss each of these boundary conditions in the following sections.

Known Incoming Surface Source

The first example of an explicit boundary condition comes from a known incoming surface source. In this case, we specify some distribution of neutrons for each $\vec{r}$ on the boundary,

$$\psi(\vec{r}, \hat{\Omega}, E, t) = \tilde{\psi}(\vec{r}, \hat{\Omega}, E, t), \quad (1.13)$$

in which $\hat{n} \cdot \hat{\Omega} < 0$ where $\hat{n}$ is the outward directed normal to the surface at the point $\vec{r}$.

Vacuum boundary conditions are a second example of explicit boundary conditions and are special case of the known incoming surface source in which $\tilde{\psi} = 0$. We will commonly use vacuum boundary conditions throughout our discussion for the remainder of this thesis.

Reflective Boundary Conditions

The only implicit boundary conditions which we will consider are reflective boundary conditions. In this case, the incoming flux from the boundary is set equal to the outgoing flux at the same location with the angle reflected [21]. Mathematically speaking, we write

$$\psi(\vec{r}, \hat{\Omega}, E, t) = \psi(\vec{r}, \hat{\Omega}', E, t), \tag{1.14}$$

in which $\vec{r}$ is on the boundary, $\hat{n} \cdot \hat{\Omega} < 0$ and

$$\hat{n} \cdot \hat{\Omega} = -\hat{n} \cdot \hat{\Omega}' \text{ and } (\hat{\Omega} \times \hat{\Omega}') \cdot \hat{n} = 0. \tag{1.15}$$

Initial Condition

For the initial condition, we must simply describe the initial distribution of neutrons in phase space,

$$\psi(\vec{r}, \hat{\Omega}, E, 0) = \tilde{\psi}(\vec{r}, \hat{\Omega}, E). \tag{1.16}$$

1.2 Discretizing the Transport Equation in Energy and Angle

In general, we cannot solve Equation 1.12 in its continuous form. For this reason, we must discretize the Equation 1.12 in time, energy, angle and space. For the majority of this thesis we will concern ourselves only with solving the time-independent problem, so let us drop the time dependence in Equation 1.12:

$$\begin{aligned}\hat{\Omega} \cdot \nabla\psi(\vec{r}, E, \hat{\Omega}) + \Sigma_t(\vec{r}, E)\psi(\vec{r}, E, \hat{\Omega}) = &\int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega})\psi(\vec{r}, E', \hat{\Omega}') \\ &+ \frac{\chi(E)}{4\pi} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \nu(\vec{r}, E')\Sigma_f(\vec{r}, E')\psi(\vec{r}, E', \hat{\Omega}) + S_{ext}(\vec{r}, \hat{\Omega}, E)\end{aligned} \tag{1.17}$$

1.2.1 Energy Discretization

We'll first discretize the Equation 1.17 in energy and, once that is complete, discretize in angle. To do so, we'll follow the method described in [21]. To begin, let's divide the energy spectrum into G energy ranges, $[E_G, E_{G-1}]$, $[E_{G-1}, E_{G-2}]$, ... , $[E_g, E_{g-1}]$, ... , $[E_1, E_0]$, where $E_G = 0$

and E_0 is to be chosen sufficiently large such that the number of particles with energies higher than E_0 may be disregarded.

We say that a neutron is in group g if its energy lies in the interval $[E_g, E_{g-1}]$. Our goal with the "multigroup" approximation is to determine the angular flux, ψ, in each of the G groups. We'll define the *group angular flux*, ψ_g, by

$$\psi_g(\vec{r}, \hat{\Omega}) = \int_{E_g}^{E_{g-1}} \psi(\vec{r}, \hat{\Omega}, E)\ dE \equiv \int_g \psi(\vec{r}, \hat{\Omega}, E)\ dE. \tag{1.18}$$

Furthermore, we wish to define the multi-group cross-sections in such a way that we preserve reaction (total, fission, scattering, absorption) rates in each group.

Now, the energy integrals in Equation 1.17 can be represented

$$\int_0^\infty \cdot\ dE' = \sum_{g'=1}^{G} \int_{g'} \cdot\ dE'.$$

With this in mind, we'll integrate Equation 1.17 over a single energy group, g, and we find

$$\begin{aligned}
&\hat{\Omega} \cdot \nabla \int_g dE\ \psi(\vec{r}, E, \hat{\Omega}) + \int_g dE\ \Sigma_t(\vec{r}, E)\psi(\vec{r}, E, \hat{\Omega}) \\
&\qquad = \sum_{g'=1}^{G} \int_g dE \int_{g'} dE' \int_{4\pi} d\hat{\Omega}'\ \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega})\psi(\vec{r}, E', \hat{\Omega}') \\
&+ \int_g dE\ \frac{\chi(E)}{4\pi} \sum_{g'=1}^{G} \int_{g'} dE' \int_{4\pi} d\hat{\Omega}'\ \ \nu(\vec{r}, E')\Sigma_f(\vec{r}, E')\psi(\vec{r}, E', \hat{\Omega}) + \int_g dE\ S_{ext}(\vec{r}, \hat{\Omega}, E)
\end{aligned} \tag{1.19}$$

As in [21], let us suppose that the group angular fluxes are separable in energy. That is,

$$\psi(\vec{r}, \hat{\Omega}, E) = f(E)\psi_g(\vec{r}, \hat{\Omega}), \tag{1.20}$$

for all $E \in [E_g, E_{g-1}]$. We interpret $f(E)$ as a normalized weighting function (a distribution of the particles throughout the energy interval) such that

$$\int_g dE\ f(E) = 1.$$

Now, we can substitute Equation 1.20 into Equation 1.19 and we obtain

$$\begin{aligned}\hat{\Omega}\cdot\nabla\psi_g(\vec{r},\hat{\Omega}) + \int_g dE\ \Sigma_t(\vec{r},E)f(E)\psi_g(\vec{r},\hat{\Omega}) \\ = \int_g dE\ S_{ext}(\vec{r},\hat{\Omega},E) + \sum_{g'=1}^{G}\int_g dE\int_{g'} dE'\int_{4\pi} d\hat{\Omega}'\ \Sigma_s(\vec{r},E',\hat{\Omega}'\to E,\hat{\Omega})f(E')\psi_g(\vec{r},\hat{\Omega}') \\ + \int_g dE\ \frac{\chi(E)}{4\pi}\sum_{g'=1}^{G}\int_{g'} dE'\int_{4\pi} d\hat{\Omega}'\ \nu(\vec{r},E')\Sigma_f(\vec{r},E')f(E')\psi_g(\vec{r},\hat{\Omega}') \quad (1.21)\end{aligned}$$

At this point, we'll begin defining the new multi-group cross-sections by

$$\Sigma_{t,g}(\vec{r}) = \int_g f(E)\Sigma_t(\vec{r},E)\ dE \quad (1.22)$$

$$\nu_g\Sigma_{f,g}(\vec{r}) = \int_g f(E)\nu(\vec{r},E')\Sigma_f(\vec{r},E)\ dE \quad (1.23)$$

$$\Sigma_{s,g'\to g}(\vec{r},\hat{\Omega}'\cdot\hat{\Omega}) = \int_g dE\int_{g'} dE' f(E')\Sigma_s(\vec{r},E'\to E,\hat{\Omega}'\cdot\hat{\Omega}), \quad (1.24)$$

the multi-group fission spectrum by

$$\chi_g = \int_g \chi(E)\ dE \quad (1.25)$$

and the group source term by

$$S_{ext,g}(\vec{r},\hat{\Omega}) = \int_g S_{ext}(\vec{r},\hat{\Omega},E)\ dE. \quad (1.26)$$

Using Equations 1.22 - 1.26, we can simplify Equation 1.21 and obtain the multi-group transport equation

$$\begin{aligned}\hat{\Omega}\cdot\nabla\psi_g(\vec{r},\hat{\Omega}) + \Sigma_{t,g}\psi_g(\vec{r},\hat{\Omega}) = S_{ext,g}(\vec{r},\hat{\Omega}) + \sum_{g'=1}^{G}\int_{4\pi} d\hat{\Omega}'\ \Sigma_{s,g'\to g}(\vec{r},\hat{\Omega}'\to\hat{\Omega})\psi_{g'}(\vec{r},\hat{\Omega}') \\ + \frac{\chi_g}{4\pi}\sum_{g'=1}^{G}\int_{4\pi} d\hat{\Omega}'\ \nu_{g'}\Sigma_{f,g'}(\vec{r})\psi_{g'}(\vec{r},\hat{\Omega}') \quad (1.27)\end{aligned}$$

1.2.2 Angular Discretization

In multiple dimensions, we begin the angular discretization process by representing the angular flux in terms of spherical harmonics

$$\psi_g(\vec{r},\hat{\Omega}) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} Y^*_{l,m}(\hat{\Omega})\phi^m_l(\vec{r}) \tag{1.28}$$

where $Y_{l,m}$ is given by [21]

$$Y_{l,m}(\hat{\Omega}) = \frac{(2l+1)(l-m)!}{(l+m)!}P^m_l(\mu)e^{i\omega m} \tag{1.29}$$

in which μ is the cosine of the polar angle, θ, and ω is the azimuthal angle from spherical coordinates. Spherical coordinates are the natural way to represent angles in three space dimensions. Here, P^m_l is the associated Legendre function defined by the formula

$$P^m_l(\mu) = (-1)^m(1-\mu^2)^{m/2}\frac{d^m}{d\mu^m}P_l(\mu) \tag{1.30}$$

in which $P^0_l(\mu) = P_l(\mu)$, the l^{th} Legendre polynomial. With this definition, we see that the spherical harmonics are orthogonal in the sense that

$$\int d\hat{\Omega}\; Y_{l,m}(\hat{\Omega})Y^*_{l',m'}(\hat{\Omega}) = 4\pi\delta_{ll'}\delta_{mm'}. \tag{1.31}$$

Furthermore, the Legendre addition theorem [21] states

$$P_l(\mu_0) = P_l(\hat{\Omega}\cdot\hat{\Omega}') = \sum_{m=-l}^{l}\frac{1}{2l+1}Y^*_{l,m}(\hat{\Omega})Y_{l,m}(\hat{\Omega}'). \tag{1.32}$$

Now, we may represent the group-to group scattering cross-section $\Sigma_{s,g'\to g}(\vec{r},\hat{\Omega}'\cdot\hat{\Omega})$ in terms of Legendre polynomials

$$\Sigma_{s,g'\to g}(\vec{r},\mu_0) = \sum_{l=0}^{\infty}\Sigma_{s,l,g'\to g}(\vec{r})P_l(\mu_0), \tag{1.33}$$

where $\mu_0 = \hat{\Omega}'\cdot\hat{\Omega}$.

Finally, we may represent the scattering term in Equation 1.27 by

$$\sum_{g'=1}^{G}\int_{g}dE\int_{g'}dE'\int_{4\pi}d\hat{\Omega}'\ \Sigma_s(\vec{r},E',\hat{\Omega}'\rightarrow E,\hat{\Omega})f(E')\psi_g(\vec{r},\hat{\Omega}')$$
$$=\sum_{l=0}^{\infty}\sum_{m=-l}^{l}Y^*_{l,m}(\hat{\Omega})\sum_{g'=1}^{G}\Sigma_{s,l,g'\rightarrow g}\phi^{g'}_{l,m}(\vec{r}) \tag{1.34}$$

where $\phi^{g'}_{l,m}$ is given by

$$\phi^{g}_{l,m}=\int_{4\pi}d\hat{\Omega}\ Y_{l,m}(\hat{\Omega})\psi_g(\vec{r},\hat{\Omega}). \tag{1.35}$$

We'll use this new representation of the scattering term for Equation 1.27 before discretizing in angle:

$$\hat{\Omega}\cdot\nabla\psi_g(\vec{r},\hat{\Omega})+\Sigma_{t,g}\psi_g(\vec{r},\hat{\Omega})=S_{ext,g}(\vec{r},\hat{\Omega})+\sum_{l=0}^{\infty}\sum_{m=-l}^{l}Y^*_{l,m}(\hat{\Omega})\sum_{g'=1}^{G}\Sigma_{s,l,g'\rightarrow g}\phi^{g'}_{l,m}(\vec{r})$$
$$+\frac{\chi_g(\vec{r})}{4\pi}\sum_{g'=1}^{G}\int_{4\pi}d\hat{\Omega}'\ \nu_g\Sigma_{f,g'}(\vec{r})\psi_{g'}(\vec{r},\hat{\Omega}). \tag{1.36}$$

At this point, we are ready to begin the angular discretization of the transport equation given by Equation 1.36. We will use the discrete ordinates (S_n) method for discretizing in angle. Using this method, we consider the transport equation evaluated only at a finite set of angles $\hat{\Omega}_n=(\mu_n,\eta_n,\xi_n)$[1] such that $\mu_n^2+\eta_n^2+\xi_n^2=1$. We wish to handle reflective boundary conditions with ease so we demand that if $\hat{\Omega}_n=(\mu_n,\eta_n,\xi_n)$ is in our quadrature set, so is $\hat{\Omega}_{n'}=(\pm\mu_n,\pm\eta_n,\pm\xi_n)$

Furthermore, we would like to retain solution invariance under 90° rotations for problems with this type of rotational symmetry. Therefore, if the angle $\hat{\Omega}=(a,b,c)$ is in our angular quadrature set, so must be (a,c,b), (b,a,c), (b,c,a), (c,a,b) and (c,b,a). With this plane- and rotational symmetry in mind, we can choose a set of N angles in the first octant and use these symmetry rules to produce the angles in the other octants.

Once our quadrature set has been chosen, we can represent the transport equation along

[1] In this definition, μ_n is not the polar angle used to define the spherical harmonics. This notation has been used to remain consistent with the popular text [21]. Relating this notation, we have $\mu_n=\sqrt{1-\mu^2}\cos(\omega)$, $\eta_n=\sqrt{1-\mu^2}\sin(\omega)$, and $\xi_n=\mu$

these $8N$ (or in 2-D, $4N$) discrete angles. This yields a system of equations,

$$\hat{\Omega}_n \cdot \nabla \psi_{g,n}(\vec{r}) + \Sigma_{t,g}\psi_{g,n}(\vec{r}) = S_{ext,g,n}(\vec{r}) + \sum_{l=0}^{\infty}\sum_{m=-l}^{l} Y^*_{l,m}(\hat{\Omega}_n)\sum_{g'=1}^{G}\Sigma_{s,l,g'\to g}\phi^{g'}_{l,m}(\vec{r}) + \frac{\chi_g}{4\pi}\sum_{g'=1}^{G}\sum_{i=1}^{8N} d\hat{\Omega}'\ \nu_g\Sigma_{f,g}(\vec{r})\psi_{g,n}(\vec{r})w_n. \quad (1.37)$$

where

$$\psi_{g,n}(\vec{r}) = \psi_g(\vec{r}, \hat{\Omega}_n) \quad (1.38)$$

and the w_n's are the associated quadrature weights.

Now, Equation 1.37 is ready for spatial discretization. This will be handled in Chapters 2 and 3 for one- and two-dimensional Cartesian geometries, respectively. In the case of isotropic scattering, Equation 1.37 simplifies dramatically:

$$\hat{\Omega}_n \cdot \nabla \psi_{g,n}(\vec{r}) + \Sigma_{t,g}\psi_{g,n}(\vec{r}) = S_{ext,g,n}(\vec{r}) + \frac{1}{4\pi}\sum_{g'=1}^{G}\Sigma_{s,g'\to g}\phi_{g'}(\vec{r}) + \frac{\chi_g(\vec{r})}{4\pi}\sum_{g'=1}^{G}\nu_{g'}\Sigma_{f,g'}(\vec{r})\phi_{g'}(\vec{r}). \quad (1.39)$$

We call $\phi(\vec{r})$ the *scalar flux.* The scalar flux is the zeroth angular moment of the angular flux, given by

$$\phi(\vec{r}) = \int_{4\pi} d\hat{\Omega}\ \psi(\vec{r}, \hat{\Omega}). \quad (1.40)$$

Furthermore, suppose that ψ depends spatially only on a single variable, z. In this case, the transport equation simplifies to the one-dimensional case

$$\mu\frac{\partial}{\partial z}\psi_g(z,\mu) + \Sigma_{t,g}\psi_g(z,\mu) = S_{ext,g}(z,\mu) + \frac{1}{2}\sum_{g'=1}^{G}\Sigma_{s,g'\to g}\phi_{g'}(z) + \frac{\chi_g}{2}\sum_{g'=1}^{G}\nu_{g'}\Sigma_{f,g'}\phi_{g'}(z). \quad (1.41)$$

We'll refer to Equation 1.41 as the one-dimensional neutron transport equation with isotropic scattering.

In the following chapters, we will discuss various methods for solving the one- and two-dimensional transport equations. In these chapters we'll describe the algorithms, discuss their merits and provide numerical results displaying solution accuracy and convergence rates. We'll look at two major classes of methods, deterministic and stochastic, before investigating the intersection of these two classes, hybrid methods.

1.3 The k-eigenvalue Problem

When analyzing a nuclear reactor it is vital to determine the criticality which we quantify using the multiplication factor, k_{eff}, of the reactor [23, 17, 21]. If $k_{eff} > 0$, the reactor is supercritical, meaning that neutrons are being produced faster than they are being destroyed. If $k_{eff} = 1$, the reactor is critical and the number of neutrons in each generation remains constant. If $k_{eff} < 0$, the reactor is subcritical.

We can compute k_{eff} by determining the largest value of k which satisfies the following problem for some non-zero angular flux:

$$\begin{aligned}\hat{\Omega} \cdot \nabla\psi(\vec{r}, E, \hat{\Omega}) + \Sigma_t(\vec{r}, E)\psi(\vec{r}, E, \hat{\Omega}) \\ = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \Sigma_s(\vec{r}, E', \hat{\Omega}' \to E, \hat{\Omega})\psi(\vec{r}, E', \hat{\Omega}') \\ + \frac{\chi(\vec{r}, E)}{4\pi k} \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \nu(\vec{r}, E')\Sigma_f(\vec{r}, E')\psi(\vec{r}, E', \hat{\Omega}). \quad (1.42)\end{aligned}$$

1.4 Organization of this Thesis

In Chapter 2, we explore methods for solving the 1-D steady-state neutron transport equation. We begin the chapter by deriving the integral transport equation before discussing the spatial discretization of the transport equation. From here, we introduce source iteration, the most simple method for solving the steady-state neutron transport equation. At this point, we'll begin discussing methods for accelerating the solution to the transport equation.

In Chapter 3, we repeat our work from Chapter 2 in 2-D. The majority of this chapter deals with the two-dimensional spatial and angular discretizations and the method for executing a 2-D transport sweep.

In Chapter 4, we consider various methods for computing the dominant eigenvalue, k_{eff}, as in Equation 1.42. Much like Chapter 2, we begin by discussing the most simple method for computing k_{eff} known as power iteration. Then, we'll look at several methods for accelerating the convergence of the eigenvalue.

In Chapter 5, we look at stochastic methods for solving the neutron transport equation. We begin with motivating the choice to incorporate Monte Carlo simulations in the solution of the transport equation. We first look at solving a 1-D steady-state transport problem using Monte Carlo methods. Next, we look at representing a transport sweep using a Monte Carlo simulation.

Chapter 6 looks at combining deterministic accelerators to accelerate Monte Carlo solutions to the neutron transport equation. We refer to these algorithms as "hybrid methods" for solving the neutron transport equation. We discuss the various challenges encountered when using

Monte Carlo transport sweeps inside deterministic accelerators and the algorithmic changes required to accommodate these stochastic function evaluations.

Chapter 2

Deterministic Solutions to the 1-D, Mono-energetic Transport Equation in Slab Geometry

In this chapter we will consider several methods for solving the one-dimensional, fixed-source neutron transport equation in slab geometry. We will begin by deriving the integral equation for 1D neutron transport. From here, we will begin the discussion of solution methods with source iteration, as it is the simplest solution method and an understanding of source iteration is prerequisite for all other methods. Once source iteration has been implemented and we are comfortable with the idea of a transport sweep, we will begin developing more advanced methods. We'll attempt to accelerate convergence using both known mathematical acceleration techniques alongside physics-based accelerators. All of the accelerators use transport sweeps at the heart of the algorithm.

2.1 Problem Definition

In this chapter we will focus our attention to solving the mono-energetic 1-D neutron transport equation in slab geometry with isotropic scattering:

$$\mu \frac{\partial \psi}{\partial x}(x,\mu) + \Sigma_t \psi(x,\mu) = \frac{1}{2}\left[\Sigma_s \phi(x) + Q(x)\right] \tag{2.1}$$

for $x \in (0,\tau)$ where $\psi(x,\mu)$ is the angular flux at the point x traveling in the direction μ. We have the following boundary conditions:

$$\psi(0,\mu) = F_0(\mu),\ \mu > 0;\ \psi(\tau,\mu) = F_\tau(\mu),\ \mu < 0 \tag{2.2}$$

These boundary conditions essentially describe the number of particles entering from the left end of the slab and the number of particles entering from the right side of the slab.

The direction cosine μ is given by $\mu = cos(\theta)$. The relationship between μ and θ can seen in Figure 2.1.

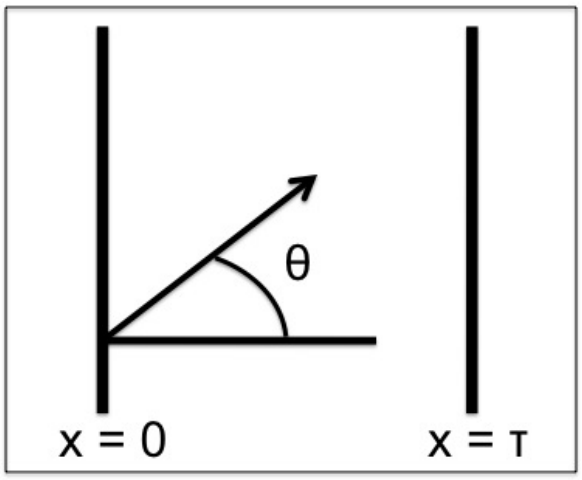

Figure 2.1: Slab of width τ

Our goal is to solve this problem in an iterative manner to a high degree of accuracy with as little computational effort as possible.

2.2 Source Iteration

In order to solve Equation 2.1 we propose an iterative scheme:

1. Begin with an initial guess $\phi_0(x)$.
2. Given $\phi_0(x)$, solve Equation 2.1 for $\psi_1(x, \mu)$.
3. Let $\phi_1(x) = \int_{-1}^{1} \psi_1(x, \mu) d\mu$.
4. Continuing this iteration until successive functions ϕ are sufficiently "close" yields a solution. That is, given $\phi_n(x)$, compute $\psi_{n+1}(x)$. Then compute $\phi_{n+1}(x) = \int_{-1}^{1} \psi_{n+1}(x, \mu) d\mu$.

2.2.1 Deriving the Integral Transport Equation

Before describing the source iteration algorithm in greater detail, let us motivate this action by deriving the integral transport equation in one-dimension. We'll perform an analysis along the same lines as found in [16]. For simplicity, let us assume that both the total and scattering cross-sections are constant.

We begin by manipulating the transport equation using a linear change of variables. Letting $z = \Sigma_t x$, we have

$$\mu\Sigma_t \frac{\partial \psi(z,\mu)}{\partial z} + \Sigma_t \psi(z,\mu) \quad = \quad \frac{1}{2}\left[\Sigma_s \phi(z) + Q(z)\right].$$

Now, dividing both sides by the total cross section, Σ_t, and letting $c = \Sigma_s/\Sigma_t$ and $S(x) = Q(x)/\Sigma_t$, we find

$$\mu \frac{\partial \psi}{\partial z} + \psi(z,\mu) \quad = \quad \frac{1}{2}\left[c\phi(z) + S(z)\right] \tag{2.3}$$

Here, since $x \in [0,\tau]$, we must have $z \in [0, X]$ where $X = \tau/\Sigma_t$.

Now, returning back to x as our spatial variable, for positive μ we compute:

$$\begin{aligned}
\mu \frac{\partial \psi}{\partial x} + \psi(x,\mu) &= \frac{1}{2}\left[c\phi(x) + S(x)\right] \\
e^{\frac{x}{\mu}}\left(\mu \frac{\partial \psi}{\partial x} + \psi(x,\mu)\right) &= e^{\frac{x}{\mu}}\frac{1}{2}\left[c\phi(x) + S(x)\right] \\
\mu \frac{\partial}{\partial x}\left[e^{\frac{x}{\mu}}\psi(x,\mu)\right] &= \frac{1}{2}e^{\frac{x}{\mu}}\left[c\phi(x) + S(x)\right] \\
\frac{\partial}{\partial x}\left[e^{\frac{x}{\mu}}\psi(x,\mu)\right] &= \frac{1}{2\mu}e^{\frac{x}{\mu}}\left[c\phi(x) + S(x)\right] \\
\int_0^y \frac{\partial}{\partial x}\left[e^{\frac{x}{\mu}}\psi(x,\mu)\right] dx &= \int_0^y \frac{1}{2\mu}e^{\frac{x}{\mu}}\left[c\phi(x) + S(x)\right] dx \\
e^{\frac{y}{\mu}}\psi(y,\mu) - \psi(0,\mu) &= \int_0^y \frac{1}{2\mu}e^{\frac{x}{\mu}}\left[c\phi(x) + S(x)\right] dx \\
\psi(y,\mu) &= e^{-\frac{y}{\mu}}\int_0^y \frac{1}{2\mu}e^{\frac{x}{\mu}}\left[c\phi(x) + S(x)\right] dx + e^{-\frac{y}{\mu}}F_0(\mu) \\
\psi(y,\mu) &= \int_0^y \frac{1}{2\mu}e^{\frac{(x-y)}{\mu}}\left[c\phi(x) + S(x)\right] dx + e^{-\frac{y}{\mu}}F_0(\mu) \\
\psi(y,\mu) &= \int_0^y \frac{1}{2\mu}e^{\frac{-|x-y|}{|\mu|}}\left[c\phi(x) + S(x)\right] dx + e^{-\frac{y}{|\mu|}}F_0(\mu)
\end{aligned}$$

For negative μ, we integrate backwards and obtain a similar expression:

$$\begin{aligned}\psi(y,\mu) &= -\int_y^X \frac{1}{2\mu} e^{\frac{(x-y)}{\mu}} \left[c\phi(x) + S(x)\right] dx + e^{\frac{(X-y)}{\mu}} F_X(\mu) \\ \psi(y,\mu) &= \int_y^X \frac{1}{2|\mu|} e^{\frac{-|x-y|}{|\mu|}} \left[c\phi(x) + S(x)\right] dx + e^{\frac{-|X-y|}{|\mu|}} F_X(\mu)\end{aligned}$$

Now, we can transform these two equations for ψ into a single relation for ϕ by noticing that

$$\begin{aligned}\phi(x) &= \int_{-1}^{1} \psi(x,\mu)\ d\mu \\ &= \int_0^1 \psi(x,\mu)\ d\mu + \int_{-1}^{0} \psi(x,\mu)\ d\mu. \end{aligned} \tag{2.4}$$

Applying Equation 2.4, we have

$$\begin{aligned}\phi(y) &= \int_0^1 \int_0^y \frac{1}{2\mu} e^{\frac{-|x-y|}{|\mu|}} \left[c\phi(x) + S(x)\right] dx d\mu + \int_0^1 e^{-\frac{y}{|\mu|}} F_0(\mu) d\mu \\ &\quad + \int_{-1}^0 \int_y^X \frac{1}{2|\mu|} e^{\frac{-|x-y|}{|\mu|}} \left[c\phi(x) + S(x)\right] dx d\mu + \int_{-1}^0 e^{\frac{-|X-y|}{|\mu|}} F_X(\mu) d\mu.\end{aligned}$$

We can split both the first and third terms in the above expression into two integrals:

$$\begin{aligned}\phi(y) &= \int_0^1 \int_0^y \frac{1}{2\mu} e^{\frac{-|x-y|}{|\mu|}} c\phi(x) dx d\mu + \int_0^1 \int_0^y \frac{1}{2\mu} e^{\frac{-|x-y|}{|\mu|}} S(x) dx d\mu \\ &\quad + \int_0^1 e^{-\frac{y}{|\mu|}} F_0(\mu) d\mu + \int_{-1}^0 \int_y^X \frac{1}{2|\mu|} e^{\frac{-|x-y|}{|\mu|}} c\phi(x) dx d\mu \\ &\quad + \int_{-1}^0 \int_y^X \frac{1}{2|\mu|} e^{\frac{-|x-y|}{|\mu|}} S(x) dx d\mu + \int_{-1}^0 e^{\frac{-|X-y|}{|\mu|}} F_X(\mu) d\mu.\end{aligned}$$

Now, we can combine the integrals containing the scalar flux and source by switching the order of integration:

$$\begin{aligned}\phi(y) &= \int_0^X \int_0^1 \frac{1}{2\eta} e^{\frac{-|x-y|}{\eta}} c\phi(x) d\eta dx + \int_0^X \int_0^1 \frac{1}{2\eta} e^{\frac{-|x-y|}{\eta}} S(x) d\eta dx \\ &\quad + \int_0^1 e^{-\frac{|y|}{|\mu|}} F_0(\mu) d\mu + \int_{-1}^0 e^{\frac{-|X-y|}{|\mu|}} F_X(\mu) d\mu.\end{aligned}$$

Finally, we rearrange terms and are left with following integral equation:

$$\phi(y) - \int_0^X k(x,y)\phi(x)\ dx = g(y)$$

where

$$\begin{aligned} k(x,y) &= \frac{c}{2}E(|x-y|) \\ E(|x-y|) &= \int_0^1 \frac{1}{\eta} e^{\frac{-|x-y|}{\eta}}\, d\eta \\ g(y) &= \int_0^X \frac{1}{2}E(|x-y|)S(x)\, dx + \int_0^1 e^{-\frac{y}{\eta}} F_0(\eta)d\eta + \int_0^1 e^{\frac{-|X-y|}{\eta}} F_X(-\eta)d\eta. \end{aligned}$$

At this point, it is easy to see that solving the transport equation is equivalent to solving the following integral equation:

$$\phi + \mathcal{K}(\phi) = g$$

where

$$\mathcal{K}(\phi(x)) = \int_0^X k(x,y)\phi(x)\, dx.$$

Now, we realize source iteration by

$$\phi^{(n+1)} = \mathcal{K}(\phi^{(n)}) + g.$$

It is important to show that source iteration converges when the domain is finite and $c < 1$. We state this as a theorem as the result provides a backbone for the remainder of this thesis.

Theorem 1. *Source iteration will converge whenever either $X < \infty$ or $c < 1$. [6]*

Proof. We wish to show that $\|\mathcal{K}\| < 1$ for some choice of operator norm. Then, by Theorem 4 (Appendix A), the iteration will converge.

We will proceed by using the 1-norm, given by

$$\|u\| = \|u\|_1 \equiv \int_0^X |u(x)|dx.$$

Now, we will prove Theorem 1 by showing that

$$\|\mathcal{K}u\| < \|u\|.$$

We have,

$$\begin{aligned}
\int_0^X |\mathcal{K}(\phi)(y)|dy &= \int_0^X \left| \int_0^X k(x,y)\phi(x)dx \right| dy \\
&\leq \int_0^X \int_0^X |k(x,y)||\phi(x)| \; dx \; dy \\
&= \int_0^X \int_0^X |k(x,y)||\phi(x)| \; dy \; dx \\
&= \int_0^X |\phi(x)| \left[\int_0^X |k(x,y)|dy \right] dx \\
&= \frac{c}{2} \int_0^X |\phi(x)| \left[\int_0^X |E_1(|x-y|)| \; dy \right] dx \\
&= \frac{c}{2} \int_0^X |\phi(x)| \left[\int_0^X E_1(|x-y|)| \; dy \right] dx \qquad (2.5) \\
&\leq \frac{c}{2} \int_0^X |\phi(x)| \left[\int_{-\infty}^{\infty} E_1(|x-y|)dy \right] dx \qquad (2.6) \\
&= \frac{c}{2} \int_0^X |\phi(x)| \left[\int_{-\infty}^{\infty} E_1(|z|)dz \right] dx \\
&= \frac{c}{2} \int_0^X |\phi(x)|dx \int_{-\infty}^{\infty} E_1(|z|)dz \\
&= \frac{c}{2} \|\phi\| \int_{-\infty}^{\infty} E_1(|z|)dz. \qquad (2.7)
\end{aligned}$$

Furthermore, we can see that

$$\begin{aligned}
\int_{-\infty}^{\infty} E_1(|z|)dz &= 2 \int_0^{\infty} E_1(y)dy \\
&= 2 \int_0^{\infty} \int_0^1 \frac{1}{\eta} e^{-y/\eta} d\eta dy \\
&= 2 \int_0^1 \frac{1}{\eta} \left[\int_0^{\infty} e^{-y/\eta} dy \right] d\eta \\
&= 2 \int_0^1 \frac{1}{\eta} \cdot \eta \; d\eta \\
&= 2 \int_0^1 1 \; d\eta \\
&= 2 \qquad (2.8)
\end{aligned}$$

Combining the results from Equations 2.7 and 2.8, we have

$$\begin{aligned} \|\mathcal{K}\phi\| &\leq \frac{c}{2}\|\phi\| \cdot \int_{-\infty}^{\infty} E_1(|z|)dz \\ &= \frac{c}{2}\|\phi\| \cdot 2 \\ &= c\|\phi\| \end{aligned}$$

Now, if $c < 1$, we have $\|\mathcal{K}\| < 1$ and the iteration converges. Furthermore, if $\phi(x)$ is non-zero everywhere, we see that the inequality between Equations 2.5 and 2.6 should be a strict inequality when $X < \infty$. Therefore, when $X < \infty$, we have $\|\mathcal{K}\| < 1$ even if $c = 1$.

□

2.2.2 Discretizing the Transport Equation

In order to solve the transport equation with source iteration, we need a method for computing $\phi^{(n+1)}$ given $\phi^{(n)}$. To do this, we must discretize the transport equation and iterate on the discrete solution. Before introducing the discretized equations, let us familiarize ourselves with the domain. Consider Figure 2.2:

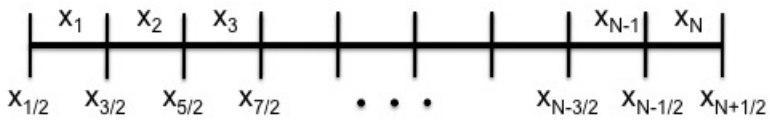

Figure 2.2: Cell faces take on half-integer indices and cell-centers take on integer indices.

In 1-D, we may choose to compute fluxes at either cell faces or cell centers. For the remainder of this chapter, let us choose to compute cell-face values of the angular flux. We'll compute the angular flux at all grid-points with half-integer values ($\psi_{i+\frac{1}{2}} = \psi(x_{i+\frac{1}{2}})$). Furthermore, we will compute the scalar flux at cell centers ($\phi_i = \phi(x_i)$). In this case, ϕ_i represents the average scalar flux over the i^{th} cell.

This choice of stencil leads to the following *diamond difference* discretization [2]:

$$\mu_k \frac{\psi^k_{i+\frac{1}{2}} - \psi^k_{i-\frac{1}{2}}}{\Delta x_i} + \Sigma_t \frac{\psi^k_{i+\frac{1}{2}} + \psi^k_{i-\frac{1}{2}}}{2} = \Sigma_s \phi_i + q_i \tag{2.9}$$

$$\phi_i = \sum_k w_k \left(\frac{\psi_{i-\frac{1}{2}} + \psi_{i+\frac{1}{2}}}{2} \right) \tag{2.10}$$

where $\psi^k_{i+\frac{1}{2}} = \psi(x_{i+\frac{1}{2}}, \mu_k)$ and $\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$. The μ_k's are chosen to be Gauss nodes and the w_k's as their corresponding weights.

For positive μ_k, we can solve Equation 2.9 for $\psi^k_{i+\frac{1}{2}}$:

$$\psi^k_{i+\frac{1}{2}} = \frac{\left(\frac{\mu^k}{\Delta x_i} - \frac{\Sigma_t}{2}\right)\psi^k_{i-\frac{1}{2}} + \Sigma_s\phi_i + q_i}{\frac{\mu^k}{\Delta x_i} + \frac{\Sigma_t}{2}} \tag{2.11}$$

Now, given an explicit boundary condition on the left boundary ($x = 0$), we can march across the domain. Similarly, for negative μ_k we can solve Equation 2.9 for $\psi^k_{i-\frac{1}{2}}$. This allows us to propagate information across the domain beginning from the right boundary ($x = X$). This idea is referred to as a "transport sweep" and can be visualized in Figure 2.3:

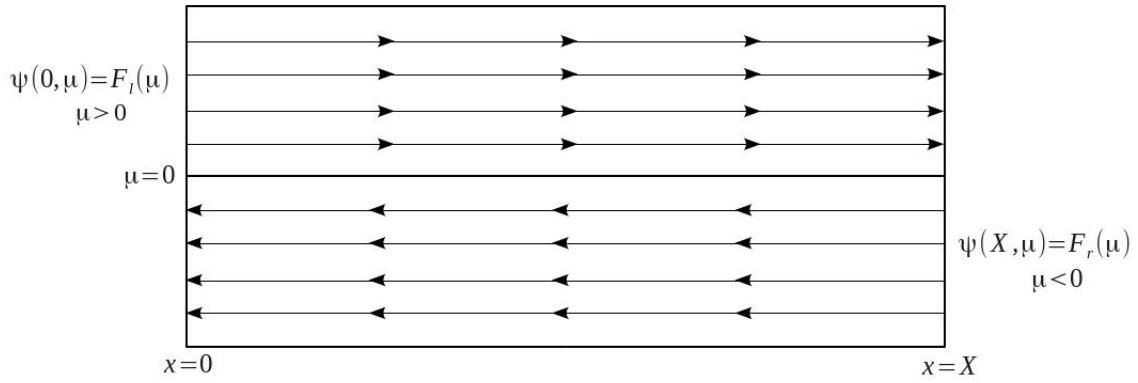

Figure 2.3: For $\mu > 0$, information propogates across the medium originating from the left boundary. For $\mu < 0$, information from the right boundary is propogated across the medium.

As we can see, as we increase the number of spatial cells and increase the number of angles in our quadrature, a transport sweep can become computationally expensive. This is not a huge issue in one-dimension, but we'll see that this problem is compounded in multiple dimensions.

We are not limited to using the previous discretization, however. In some cases, it may be advantageous to compute both the angular and scalar fluxes at cell faces. The choice of which discretization we use is often dependent on what type of accelerations schemes we would like to use. These will be discussed in further detail in the upcoming sections.

2.2.3 Analyzing Source Iteration

Source iteration converges whenever the width of the slab is finite and/or the ratio $\Sigma_s/\Sigma_t < 1$, however we are not guaranteed that source iteration will converge in a timely manner. When $\Sigma_s/\Sigma_t \approx 1$ and the mean-free-path $(1/\Sigma_t)$ of a neutron is far shorter than the width of the slab, convergence may take hundreds or thousands of iterations to converge [2]. Since each iteration is on the order of $n \cdot k$ work, this can be quite expensive, and recall, this is for a 1-D calculation. Each iteration will become more expensive as we reach higher dimensions, add in multiple energy groups and assume anisotropic scattering.

The following theorem helps us to quantify how poorly source iteration may perform in certain problem regimes.

Theorem 2. *Suppose Equation 2.1 for $x \in [-\infty, \infty]$ has a solution $\psi^*(x,\mu)$. Define an error function, $e^{(k)}(x,\mu) = \psi^*(x,\mu) - \psi^{(k)}(x,\mu)$ where the $\psi^{(k)}$ are the sequence of iterates produced by source iteration. Then we have*

$$\left| e^{(k)}(x,\mu) \right| \leq B \cdot c^k \tag{2.12}$$

where B is a constant determined by the initial error and $c = \Sigma_s/\Sigma_t$ is the scattering ratio.

Proof. We'll prove this theorem using the method described in [2]. Consider the exact transport equation (Equation 2.13) and the source iteration scheme (Equation 2.14) for an infinite medium ($x \in [-\infty, \infty]$),

$$\mu \frac{\partial \psi^*}{\partial x}(x,\mu) + \Sigma_t \psi^*(x,\mu) = \frac{1}{2}\left[\Sigma_s \int_{-1}^{1} \psi^*(x,\mu')d\mu' + Q(x)\right] \tag{2.13}$$

$$\mu \frac{\partial \psi^{(k+1)}}{\partial x}(x,\mu) + \Sigma_t \psi^{(k+1)}(x,\mu) = \frac{1}{2}\left[\Sigma_s \int_{-1}^{1} \psi^{(k)}(x,\mu')d\mu' + Q(x)\right]. \tag{2.14}$$

Subtracting Equation 2.14 from 2.13 yields the following equation for the error

$$\mu \frac{\partial e^{(k+1)}}{\partial x}(x,\mu) + \Sigma_t e^{(k+1)}(x,\mu) = \frac{1}{2}\left[\Sigma_s \int_{-1}^{1} e^{(k)}(x,\mu')d\mu'\right]. \tag{2.15}$$

Equation 2.15 relates the current error to the previous error, and iteratively, the current error to the initial error.

Now, consider expanding the k^{th} error, $e^{(k)}$, as a Fourier integral,

$$e^{(k)}(x,\mu) = \int_{-\infty}^{\infty} A^{(k)}(\lambda,\mu) e^{i\Sigma_t \lambda x} d\lambda. \tag{2.16}$$

Substituting Equation 2.16 into Equation 2.15, we have

$$\int_{-\infty}^{\infty} \left[\Sigma_t (i\lambda\mu + 1) A^{(k+1)}(\lambda, \mu) - \frac{\Sigma_s}{2} \int_{-1}^{1} A^{(k)}(\lambda, \mu') d\mu' \right] e^{i\Sigma_t \lambda x} d\lambda = 0. \tag{2.17}$$

Therefore, we must have

$$A^{(k+1)}(\lambda, \mu) = \frac{c}{2} \frac{1}{1 + i\lambda\mu} \int_{-1}^{1} A^{(k)}(\lambda, \mu') d\mu' \tag{2.18}$$

for $\lambda \in \mathbb{R}$ and for $\mu \in [-1, 1]$. We assume that $c < 1$ so that Theorem 1 guarantees convergence.

Integrating Equation 2.18 over all μ yields

$$\int_{-1}^{1} A^{(k+1)}(\lambda, \mu') d\mu' = \omega(\lambda) \int_{-1}^{1} A^{(k)}(\lambda, \mu') d\mu' \tag{2.19}$$

in which

$$\begin{aligned} \omega(\lambda) &\equiv \frac{c}{2} \int_{-1}^{1} \frac{1}{1 + i\lambda\mu} d\mu & (2.20) \\ &= c \int_{0}^{1} \frac{1}{1 + \lambda^2 \mu^2} d\mu & (2.21) \\ &= \frac{c}{\lambda} \tan^{-1} \lambda. & (2.22) \end{aligned}$$

Then, for all $k \geq 0$, Equation 2.19 implies

$$\int_{-1}^{1} A^{(k)}(\lambda, \mu') d\mu' = \omega(\lambda)^k \int_{-1}^{1} A^{(0)}(\lambda, \mu') d\mu' \tag{2.23}$$

Applying Equation 2.18 to the left-hand side of Equation 2.23 yields

$$A^{(k+1)}(\lambda, \mu) = \frac{c}{2} \frac{\omega^k}{1 + i\lambda\mu} \left(\int_{-1}^{1} A^{(0)}(\lambda, \mu') d\mu' \right) \tag{2.24}$$

Now, we have an explicit formula for the error based on the initial error for all $k \geq 0$, as given by

$$e^{(k+1)}(x, \mu) = \int_{-\infty}^{\infty} \frac{c}{2} \frac{\omega^k}{1 + i\lambda\mu} \left(\int_{-1}^{1} A^{(0)}(\lambda, \mu') d\mu' \right) e^{i\Sigma_t \lambda x} d\lambda. \tag{2.25}$$

We can bound the $(k+1)^{st}$ error as follows,

$$\begin{aligned}\left|e^{(k+1)}(x,\mu)\right| &\leq \frac{c}{2}\int_{-\infty}^{\infty}|\omega(\lambda)|^k\left|\int_{-1}^{1}A^{(0)}(\lambda,\mu')d\mu'\right|d\lambda \\ &= \left[\frac{c}{2}\int_{-\infty}^{\infty}\left|\int_{-1}^{1}A^{(0)}(\lambda,\mu')d\mu'\right|d\lambda\right]\sigma_{SI}^k \\ &= B\cdot\sigma_{SI}^k, \end{aligned} \tag{2.26}$$

where B is a constant based on the Fourier expansion coefficient of the initial error and spectral radius of the source iterations scheme is given by

$$\sigma_{SI} = \max_{\lambda\in\mathbb{R}}\omega(\lambda) = \max_{\lambda\in\mathbb{R}}\left(\frac{c}{\lambda}\tan^{-1}\lambda\right) = \omega(0) = c. \tag{2.27}$$

□

Clearly, as $c \to 1$ and in the case of an infinite medium, the iteration may become arbitrarily slow. When $c = .99$, it will take over 200 iterations to decrease the error by a single order of magnitude. When $c = .999$, over 2000 iterations will be required to decrease the error by an order of magnitude. This helps motivate us to put forth effort in developing efficient and robust accelerators to solve the neutron transport equation.

Before moving on, we can perform further analysis of the source iteration scheme that will aid us in building these accelerators. As in [2], let us assume that the initial error consists of a single Fourier mode,

$$e^{(0)}(x,\mu) = a(\mu)e^{-\Sigma_t\lambda x}. \tag{2.28}$$

Then, for all iterates $k \geq 0$, the error is comprised of this single Fourier mode and the above derivation demonstrates

$$e^{(k+1)}(x,\mu) = \omega^k(\lambda)\left[\frac{c}{2}\frac{1}{1+i\lambda\mu}\left(\int_{-1}^{1}a(\mu')d\mu'\right)\right]e^{i\Sigma_t\lambda x}. \tag{2.29}$$

Source iteration will converge slowest for modes in which λ is nearly 0. In other words, the slowest converging components of the solution have weak spatial dependence, or are the long-wavelength components of the solution [2].

We can demonstrate this numerically with the following example. We have computed the solution to a one-dimensional, single-speed transport equation in homogenous slab geometry ($\Sigma_s = 9.9$, $\Sigma_t = 10$, $\tau = 10$) using 100 grid points. Call this solution ϕ^*. Now, let us perturb

this solution,

$$\phi^{(0)} = \phi^*(x) + .01 \sin\left(\frac{\pi x}{\tau}\right) + .01 \sin\left(\frac{40\pi x}{\tau}\right).$$

Figure 2.5 shows the initial error along with the error after 25, 50, and 100 iterations.

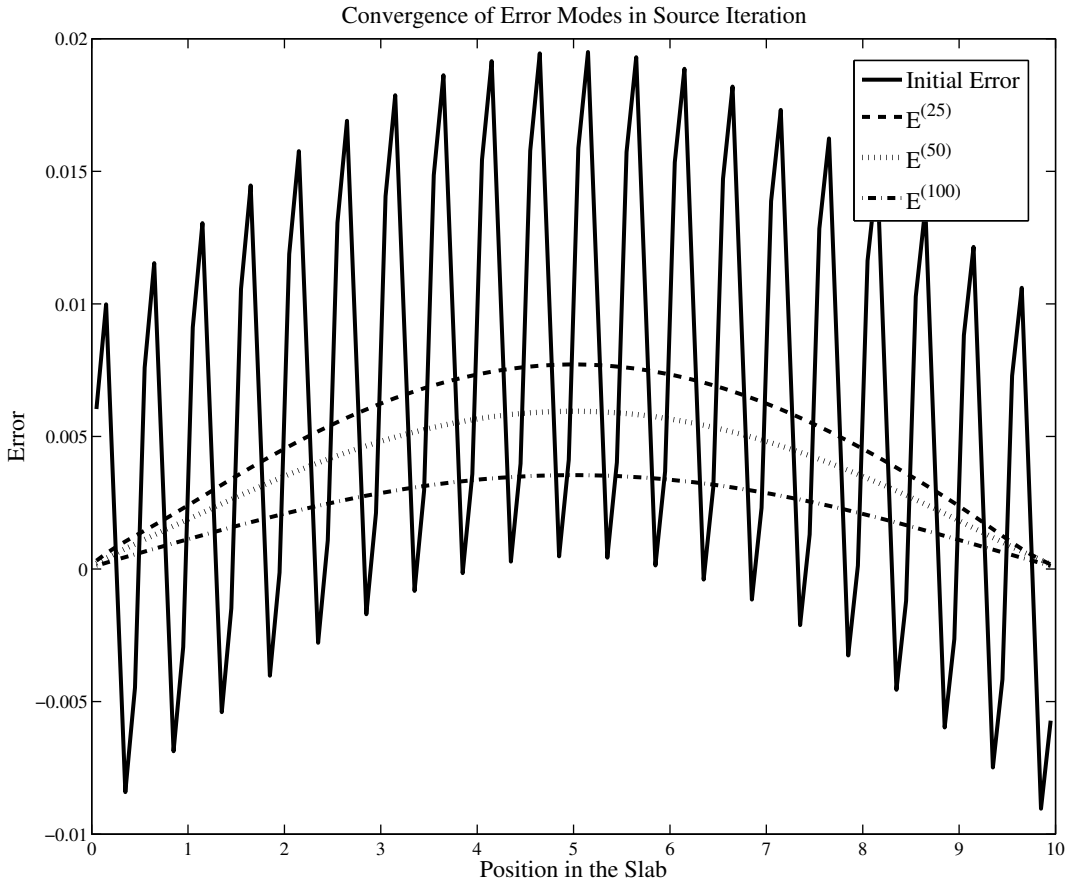

Figure 2.4: We see that the long wavelength error mode converges at a much slower rate than the short wavelength error mode.

After only 25 iterations the high-frequency error mode as been nearly completely damped, however the long-wavelength mode still persists even after 100 iterations. This analysis suggests that any numerical acceleration techniques should work to resolve the long-wavelength modes

of the solution, as standard source iterations will quickly resolve the high-frequency modes, while simultaneously ensuring that high-frequency errors are not amplified.

2.3 Using Krylov Methods to Solve the Transport Equation

In general, source iteration is not a feasible method for solving the neutron transport equation. Even for relatively simple problems, one energy group and 1-D, these problems can require thousands of iterations in order to converge to the desired tolerance. For this reason, we need to employ more advanced methods to cut down on the amount of work required to obtain a solution. In the nuclear engineering community, algorithms which attempt to cut down the number of transport sweeps required to converge are often referred to as "accelerators."

In order to utilize Krylov methods to solve this problem, we need to represent this problem as a linear system of equations. Early in this chapter we showed that we could represent source iteration using operator notation as:

$$\phi^{(n+1)} = \mathcal{K}(\phi^{(n)}) + g \tag{2.30}$$

where $\mathcal{K}$ is a linear operator.

Equation 2.30 has converged if $\phi^{(n)} = \phi^{(n+1)}$. Instead of using source (or Picard) iteration to solve this problem, we should be able to increase the rate of convergence by using Krylov methods. In this case, we'll describe the problem slightly differently, writing the integral equation as:

$$\phi(x) - \mathcal{K}(\phi)(x) = g(x) \tag{2.31}$$

$$(I - \mathcal{K})\phi(x) = g(x) \tag{2.32}$$

where I is the identity function.

Once we discretize, this becomes a linear system of equations

$$A\vec{\phi} = \vec{g}. \tag{2.33}$$

where A comes from the discretization of $I - \mathcal{K}$.

It is important to note that even though we can abstractly represent this problem as a linear system of equations, we never form the matrix A. In many cases, computing the coefficient matrix A would require more numerical work than the solution to the transport equation. Furthermore, for the Krylov methods which we will employ, a representation of the matrix is not necessary. Instead, we must only be able to write a formula (or routine) to compute the action of the matrix A on the vector ϕ.

2.3.1 GMRES

One of the most common Krylov methods is GMRES, which stands for Generalized Minimum RESidual. When solving $A\vec{x} = \vec{b}$, GMRES minimizes $||\vec{b} - A\vec{x}||_2$ over the space $x_0 + K_k$, where K_k denotes the k^{th} Krylov subspace [15]. This is defined as

$$K_k = \text{span}(r_0, Ar_0, ...A^{k-1}r_0) \tag{2.34}$$

GMRES is guaranteed (in the absence of roundoff error) to converge in n or less iterations when the dimension of the problem is n (i.e. x is a vector of length n). We prove this fact in Appendix A.

For well-conditioned problems (problems where $K(A) = ||A|| \cdot ||A^{-1}|| \approx small$), GMRES can converge quite quickly. The convergence behavior is determined by the spectrum of A, or the set of eigenvalues of A. It can be shown ([1]) that GMRES performs best when these eigenvalues are grouped in a few tight clusters.

The other selling point for GMRES is that we do *not* actually need to form the matrix A. Instead, we only need a matrix-vector-product (hereafter referred to as a "matvec"), that is, a routine that can compute the action of A on a vector. For the neutron transport problem, creating the actual matrix could require far too much work to be a reasonable solution, so we can take advantage of this property of GMRES. A more detailed discussion of GMRES can be found in Appendix A.

2.3.2 Building the Matrix-Vector-Product

For the neutron transport problem, our matvec is straightforward to build. Our matvec, M, must represent the action of $(I - \mathcal{K})$ on a vector. The matrix, M, is simply the discrete version of the operator $(I - \mathcal{K})$.

If we recall our discussion of source iteration, we have a mapping that takes ϕ^l to ϕ^{l+1}. We can represent this as $G(\phi^l) = \phi^{l+1}$. We've found our solution when $G(\phi) = \phi$. Furthermore, in the previous section, we found that $\phi = \mathcal{K}\phi + b$. Combining these two facts, we can see that

$$G(\phi) = \mathcal{K}\phi + b. \tag{2.35}$$

Therefore, the action of A on a vector can be represented as $\mathcal{K}\phi = G(\phi) - b$. If we need to compute b, we can see that $b = G(\vec{0})$.

Now, our matvec, M, can be explicitly defined:

$$M\phi = (I - \mathcal{K})\phi = \phi - (G(\phi) - b) = \phi - G(\phi) + b = \phi - G(\phi) + G(\vec{0}) \tag{2.36}$$

Now, we ask GMRES to solve the problem $M\vec{\phi} = \vec{b}$. We'll briefly note that formulating the problem in this way is a good choice. GMRES operates on vectors that are the same dimension as ϕ, not the dimension of ψ. When n (number of spatial cells) and k (number of angular cells) are large, this choice can save us a great deal of computational work and memory.

2.3.3 Other Krylov Methods

It is important to note that GMRES is not our only choice of Krylov methods for solving this problem. For large problems, GMRES may not be an option due to memory concerns. Recall, for every iteration, GMRES must store an additional Krylov vector as demonstrated in Equation 2.34. If a problem requires many iterations, memory limits may hinder our ability to use GMRES. This leaves us with two options: GMRES(m) or low-storage Krylov methods.

GMRES(m), or GMRES with restarts, attempts to remedy the issue of limited memory by storing a maximum of m Krylov vectors. After m iterations, we check to see if the solution has converged. If it has not, we take the current solution and use it as an intial guess for the next pass of GMRES.

This will generally slow down convergence since we are throwing away all of the information that we have built up in our Krylov basis in order to make room for more Krylov vectors. However, this helps us get around memory issues that plagued us when storing the full Krylov basis.

Our other option is a low-storage Krylov method. Two examples of these are TFQMR and BiCGSTAB. The storage for both of these algorithms is bounded throughout the linear iteration. However, each of these methods come with some inherent danger. Both of these methods are subject to break down within the iteration, generally due to a division by zero [15]. Furthermore, the convergence theory for these two methods is far less developed that that of GMRES. For this reason, we usually only turn to these methods when issues arise with GMRES. In practice, for solving these neutron transport problems, both TFQMR and BiCGSTAB generally require more function evaluations (transport sweeps) than GMRES in order to meet the convergence criteria.

2.4 Anderson Acceleration

Anderson Acceleration [29] was originally formulated by D.G. Anderson in the mid 1960s. In general, a fixed-point iteration solves the problem $x = G(x)$ using the following steps:

- Start with initial iterate x_0
- $x_{k+1} = G(x_k)$, until convergence.

While this algorithm is very simple, convergence may be very slow for some problems [29]. Many problems of the form $x = G(x)$, where G is a nonlinear map may be recast in the form $F(x) = x - G(x)$ and solved with Newton's method. In some situations, this may be a huge advantage. In other situations, it may be beneficial to avoid the root-finding formulation and keep the problem in fixed-point form [29]. In this case, one can attempt to increase convergence rates using Anderson Acceleration:

Anderson Acceleration(m) - m vector storage

Given x_0 and $m \geq 1$.
Compute $x_1 = G(x_0)$.
for k = 1, 2, ... **do**
Set $m_k = \min\{m, k\}$.
Set $F_k = (f_{k-m_k}, ..., f_k)$, where $f_i = G(x_i) - x_i$.
Compute $\alpha^{(k)} = (\alpha_0^{(k)}, ..., \alpha_{m_k}^{(k)})^T$ where

$$\alpha^{(k)} = \text{argmin}_\alpha \|F_k \alpha\|_2 \text{ s.t. } \sum_{i=0}^{m_k} \alpha_i = 1 \tag{2.37}$$

Set $x_{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i})$
end for

In general, we may choose to keep the number of vectors, m, that we store small for two reasons:

1. These vectors may be quite large and storage may become a concern.
2. The minimization 2.37 may become expensive to solve or the problem may become ill-conditioned.

It has been shown that in the case of a linear function G, Anderson Acceleration behaves much like GMRES [29], but in the case of nonlinear functions, much less theory exists.

In the case of neutron transport, source iteration can be represented as a fixed-point problem, $\phi = G(\phi)$. We can apply Anderson Acceleration to achieve faster convergence in many regimes. Anderson Acceleration, or the intimately related accelerator, Nonlinear Krylov Acceleration, has been applied to the k-eigenvalue problem in [4].

2.5 High-Order/Low-Order Accelerators

One of the major ways nuclear engineers have tried to speed up the solution to the neutron transport equation is by coupling the neutron transport equation to the closely related diffusion equation. It can be shown that for highly scattering media (i.e. materials where $\Sigma_s/\Sigma_t \approx 1$), that diffusion models can approximate transport in some regions of the domain. Luckily for us, the diffusion equation is very easy to solve, especially in 1-D where solving the diffusion equation requires only a tridiagonal matrix solve.

Since the 1960's and 1970's it has been a trend in the nuclear engineering community to couple the high-order neutron transport equation to a low-order diffusion equation in such a way as to minimize the number of transport sweeps required to converge to a solution [2, 14]. The idea is that if we can use the diffusion equation to gain better approximations, we are willing to invest some time in solving the easy-to-solve diffusion problem if it means we will spend less time executing transport sweeps. The two equations are coupled in such a way that when the transport equation is solved, the diffusion equation is satisfied simultaneously.

There are many models and variants of this process available, but we will primarily focus on Quasi-Diffusion (QD) [14, 2] and Nonlinear Diffusion Acceleration (NDA) as described in [18]. Quasi-Diffusion has a long history and the ideas that were developed for quasi-diffusion are the basis for many of these High-order/Low-order solvers. I will describe the process of nonlinear diffusion acceleration (NDA) and show that when the high-order (HO) problem is solved, then necessarily the low-order (LO) problem is solved; that is, if ϕ^{HO} is the solution to the high order problem and ϕ^{LO} is the solution to the low order problem, then upon convergence, $\phi^{HO} = \phi^{LO}$. In general, achieving consistency between the HO and LO problem has been troublesome and has been called "the tyranny of consistent low-order discretization" [2]. This nonlinear problem is traditionally solved using Picard iteration or successive substitution, but we can speed up this process by using a Jacobian-Free Newton-Krylov method as a nonlinear solver. In this case, we will refer to the method as JFNK-NDA.

2.5.1 Nonlinear Diffusion Acceleration (NDA)

In this description of NDA, we will follow closely the development described in [18], filling in gaps and providing more detail when necessary. Before continuing, we need to define the current, J, as follows

$$J(x) = \int_{-1}^{1} \mu\psi(x,\mu)d\mu \tag{2.38}$$

and we should recall that

$$\phi(x) = \int_{-1}^{1} \psi(x,\mu) d\mu. \tag{2.39}$$

Now, let us compute the 0^{th} moment of the transport equation:

$$\begin{aligned}
\int_{-1}^{1} \left[\mu \frac{\partial}{\partial x}\psi(x,\mu) + \Sigma_t \psi(x,\mu)\right] d\mu &= \int_{-1}^{1} \left[\frac{1}{2}\left(\Sigma_s \phi(x) + Q(x)\right)\right] d\mu \\
\int_{-1}^{1} \mu \frac{\partial}{\partial x}\psi(x,\mu) d\mu + \Sigma_t \int_{-1}^{1} \psi(x,\mu) d\mu &= \int_{-1}^{1} \left[\frac{1}{2}\left(\Sigma_s \phi(x) + Q(x)\right)\right] d\mu \\
\frac{\partial}{\partial x}\int_{-1}^{1} \mu\psi(x,\mu) d\mu + \Sigma_t \int_{-1}^{1} \psi(x,\mu) d\mu &= \int_{-1}^{1} \left[\frac{1}{2}\left(\Sigma_s \phi(x) + Q(x)\right)\right] d\mu \\
\frac{\partial}{\partial x} J(x) + \Sigma_t \phi(x) &= \Sigma_s \phi(x) + Q(x) \\
\frac{\partial}{\partial x} J(x) + \left(\Sigma_t - \Sigma_s\right)\phi(x) &= Q(x)
\end{aligned} \tag{2.40}$$

We can rewrite Eq. 2.40 as

$$\frac{dJ}{dx} + (\Sigma_t - \Sigma_s)\phi = Q. \tag{2.41}$$

Now, as in [18], we formulate the current as a combination of the Fick's law diffusion approximation alongside a drift term

$$J = -\frac{1}{3\Sigma_t}\frac{d\phi}{dx} + \hat{D}\phi \tag{2.42}$$

where $\hat{D}(x)$ is defined in such a way as to ensure consistency between the high-order and low-order problem. $\hat{D}$ is computed by using high-order scalar fluxes and currents that come directly from taking moments of the high-order angular flux.

Combining Eq. 2.42 and Eq. 2.41 yields the Low-Order Problem for NDA:

$$\frac{d}{dx}\left[-\frac{1}{3\Sigma_t}\frac{d\phi}{dx} + \hat{D}\phi\right] + \left(\Sigma_t - \Sigma_s\right)\phi = Q. \tag{2.43}$$

Discretizing the NDA Equations

In order to implement NDA, we need to discretize Eqs. 2.42 and 2.43. Discretizing Eq. 2.42 yields

$$J^{HO}_{i+\frac{1}{2}} = -\frac{1}{3\Sigma_t}\frac{\phi^{HO}_{i+1} - \phi^{HO}_{i}}{\Delta x} + \frac{\hat{D}_{i+\frac{1}{2}}}{2}\left(\phi^{HO}_{i+1} + \phi^{HO}_{i}.\right) \tag{2.44}$$

We use Eq 2.44 to define the $\hat{D}$ term. We notice that all other terms in Eq 2.44 are comprised of high-order components. This is what guarantees the consistency between the high- and low-order problems. Solving for the $\hat{D}$ term gives us

$$\hat{D}_{i+\frac{1}{2}} = \frac{J^{HO}_{i+\frac{1}{2}} + \frac{1}{3\Sigma_t}\left(\frac{\phi^{HO}_{i+1}-\phi^{HO}_{i}}{\Delta x}\right)}{\frac{\phi^{HO}_{i+1}+\phi^{HO}_{i}}{2}}. \tag{2.45}$$

Now, we are ready to discretize the low-order equation, Eq. 2.43, and we get

$$-\frac{1}{3\Sigma_t}\frac{\phi_{i+1}-2\phi_i+\phi_{i-1}}{\Delta x^2} + \frac{\hat{D}_{i+\frac{1}{2}}}{2\Delta x}\left(\phi_{i+1}+\phi_i\right) - \frac{\hat{D}_{i-\frac{1}{2}}}{2\Delta x}\left(\phi_i+\phi_{i-1}\right) + \left(\Sigma_t-\Sigma_s\right)\phi_i = Q_i. \tag{2.46}$$

Consistency of HO and LO Equations

Suppose we have found a solution to the high-order problem, $\phi^* = (\phi^*_1 \cdots \phi^*_n)^T$. Then, we know that Eq. 2.9 is satisfied. That is, we have

$$\mu^k \frac{\psi^k_{i+\frac{1}{2}} - \psi^k_{i-\frac{1}{2}}}{\Delta x} + \Sigma_t \frac{\psi^k_{i+\frac{1}{2}} + \psi^k_{i-\frac{1}{2}}}{2} \quad = \quad \frac{1}{2}\Sigma_s\phi^*_i + \frac{1}{2}Q_i \tag{2.47}$$

where ψ must also be the true ψ since it is computed using ϕ^*.

We sum both sides over all μ^k to find that we must have

$$\frac{J_{i+\frac{1}{2}} - J_{i-\frac{1}{2}}}{\Delta x} + \frac{\Sigma_t}{2}\phi^*_i = \Sigma_s\phi^*_i + Q_i \tag{2.48}$$

where the right hand side is simply multiplied by 2 since each term is independent of μ and $\sum_k w^k = 2$.

Now, let us substitute this solution into Eq. 2.46. This yields

$$-\frac{1}{3\Sigma_t}\frac{\phi^*_{i+1}-2\phi^*_i+\phi^*_{i-1}}{\Delta x^2} + \frac{\hat{D}_{i+\frac{1}{2}}}{2\Delta x}\left(\phi^*_{i+1}+\phi^*_i\right) - \frac{\hat{D}_{i-\frac{1}{2}}}{2\Delta x}\left(\phi^*_i+\phi^*_{i-1}\right) + \left(\Sigma_t-\Sigma_s\right)\phi^*_i = Q_i.$$

Substituting in the definition of $\hat{D}$ gives

$$-\frac{1}{3\Sigma_t}\frac{\phi^*_{i+1}-2\phi^*_i+\phi^*_{i-1}}{\Delta x^2} + \frac{J_{i+\frac{1}{2}}}{\Delta x} + \frac{1}{3\Sigma_t}\frac{\phi^*_{i+1}-\phi^*_i}{\Delta x^2} - \frac{J_{i-\frac{1}{2}}}{\Delta x} - \frac{1}{3\Sigma_t}\frac{\phi^*_i-\phi^*_{i-1}}{\Delta x^2} + \left(\Sigma_t-\Sigma_s\right)\phi^*_i = Q_i.$$

Combining the $\frac{J}{\Delta x}$ terms and canceling ϕ^*'s leaves us with exactly Equation 2.48, thus showing the high-order and low-order problems are consistent.

We also must derive boundary conditions for the low-order problem. In the case of a

reflective boundary, we must enforce that the derivative of the scalar flux is zero at the boundary. Vacuum boundaries are handled in a slightly more sophisticated manner.

For vacuum boundaries, we proceed as in [18]. We begin by enforcing consistency between the high-order and low-order problems at the boundary,

$$\frac{J^{LO}_{\frac{1}{2}}}{\phi^{LO}_1} = \frac{J^{HO}_{\frac{1}{2}}}{\phi^{HO}_1}. \tag{2.49}$$

We continue by discretizing the neutron balance equation, the 0^{th} angular moment of the neutron transport equation. This yields

$$\frac{J^{LO}_{\frac{3}{2}} - J^{LO}_{\frac{1}{2}}}{\Delta x} + (\Sigma_t - \Sigma_s)\phi^{LO}_1 = Q_1. \tag{2.50}$$

Now, we use Equation 2.44 to compute an expression for $J^{LO}_{\frac{3}{2}}$,

$$J^{LO}_{\frac{3}{2}} = -\frac{1}{3\Sigma_t}\frac{\phi^{LO}_2 - \phi^{LO}_1}{\Delta x} + \frac{\hat{D}_{\frac{3}{2}}}{2}\left(\phi^{LO}_2 + \phi^{LO}_1.\right) \tag{2.51}$$

We can write $J^{LO}_{\frac{1}{2}}$ in terms of ϕ^{LO}_1 using Equation 2.49,

$$J^{LO}_{\frac{1}{2}} = \frac{J^{HO}_{\frac{1}{2}}}{\phi^{HO}_1}\phi^{LO}_1. \tag{2.52}$$

Substituting Equations 2.51 and 2.52 into Equation 2.50, yields a single linear equation with unknowns ϕ^{LO}_1 and ϕ^{LO}_2. The high-order solution provides $\hat{D}$, $J^{HO}_{\frac{1}{2}}$, and ϕ^{HO}_1.

Implementing Picard NDA

In the most simple implementation of NDA, we solve alternately the high-order and low-order equations. We iterate according to the process described in [18]:

1. Given a scalar flux, ϕ^n, execute a single transport sweep to yield $\psi^{n+\frac{1}{2}}$.
2. Take the new angular flux, $\psi^{n+\frac{1}{2}}$, and compute $\hat{D}^{n+\frac{1}{2}}$ and the boundary conditions.
3. Solve the low-order problem, 2.46, for ϕ^{n+1}.
4. Repeat until convergence.

We can represent this process as a new fixed-point problem, $\phi = G_{NDA}(\phi)$. In practice, we see that Picard NDA is an vast improvement over source iteration when the medium is highly

scattering. This is demonstrated in the results section at the end of this chapter. According to [18], Picard-NDA may encounter trouble when the optical depth of a cell, $\Delta x \cdot \Sigma_t$, is large and may fail to converge. We have found that applying Anderson Acceleration to the fixed-point problem can increase the rate of convergence and make the algorithm more robust. However, in practice we may wish to abandon Picard-NDA and instead turn to a more robust method of implementation; we utilize a Jacobian-Free Newton-Krylov method to solve this coupled high-order/low-order system which we will discuss in the next section.

Analyzing Picard-NDA

In Section 2.2.3, we observed that source iteration damped high-frequency error modes quickly, whereas the long-wavelength error mode took hundreds of iterations to resolve. Let us consider a similar example. We'll use the same perturbation to the exact solution,

$$\phi^{(0)} = \phi^*(x) + .01 \sin\left(\frac{\pi x}{\tau}\right) + .01 \sin\left(\frac{40\pi x}{\tau}\right),$$

and observe how these error modes are eliminated using Picard NDA.

Figure 2.5 shows the initial error along with the error after 1 and 10 iterations.

As we can see, after a single iteration, the high-frequency error mode has been damped considerably. Furthermore, the long-wavelength error mode has been nearly completely eliminated. Using the diffusion equation as a low-order equation effectively removes these long-wavelength error modes.

Implementing JFNK-NDA

In order to use Newton's method to solve this nonlinear problem, we need to pose this question in a different form. We need to create a function, F, which takes as input ϕ and is identically zero when ϕ satisfies the transport equation. The computations done here are similar to that of the iteration process for Picard NDA, but are changed slightly as to couple the high-order and low-order equations. Once we build this function, $F(\phi)$, we will feed it to the Newton-Krylov code which will solve $F(\phi) = 0$.

The function F is built as follows, as described in [18]:

1. Begin by inputting ϕ, a vector, which is a descretized version of $\phi(x)$.

2. Execute a transport sweep using ϕ taken as the input for F. The result of this transport sweep is ψ^{HO}. The transport sweep is accomplished using the discretized transport equation, Equation 2.9.

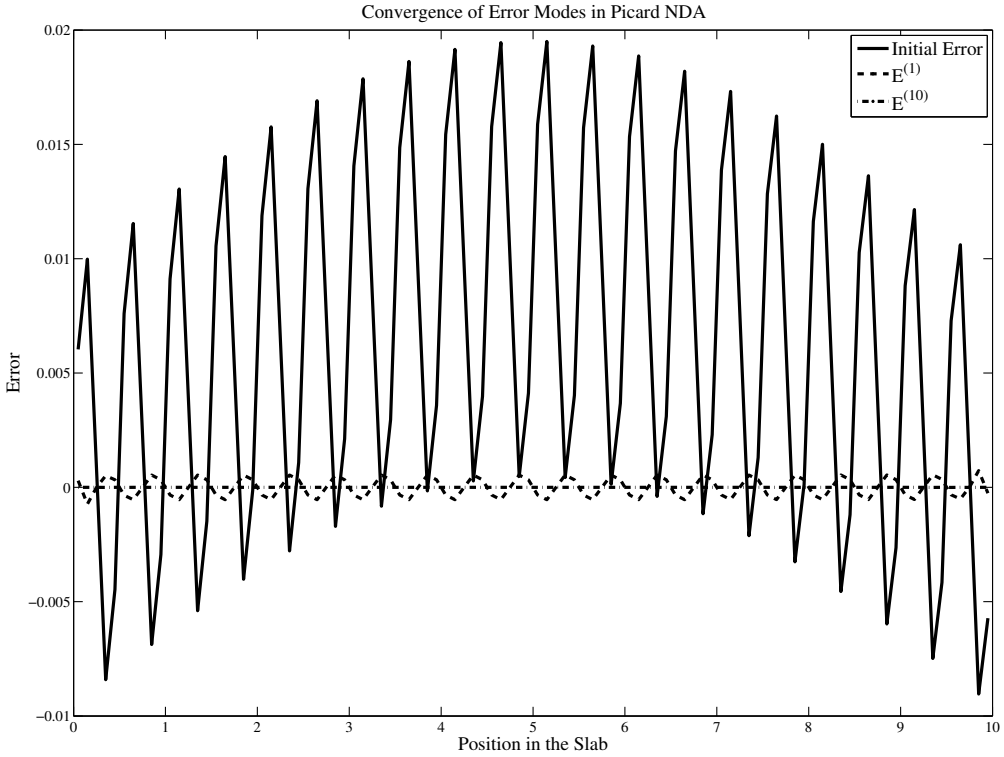

Figure 2.5: We see that both short- and long-wavelength error modes are damped quickly using Picard NDA.

3. Given ψ^{HO} we can compute $\hat{D}$. This is accomplished by solving the following equation for $\hat{D}$:

$$J^{HO}_{i+\frac{1}{2}} = -\frac{1}{3\Sigma_t}\frac{\phi^{HO}_{i+1} - \phi^{HO}_i}{\Delta x} + \frac{\hat{D}_{i+\frac{1}{2}}}{2}\left(\phi^{HO}_{i+1} + \phi^{HO}_i.\right) \tag{2.53}$$

In order to solve for $\hat{D}$, we will first need to compute two other values:

$$J^{HO}_{i+\frac{1}{2}} = \int_{-1}^{1} \mu\psi^{HO}_{i+\frac{1}{2}} d\mu \tag{2.54}$$

$$\phi^{HO}_i = \frac{1}{2}\int_{-1}^{1}\left(\psi^{HO}_{i+\frac{1}{2}} + \psi^{HO}_{i-\frac{1}{2}}\right) d\mu \tag{2.55}$$

We notice here that due to its construction, if ϕ has length n, $\hat{D}$ must have length $n-1$. Therefore, $\hat{D}$ is computed at each cell face, except for the left and right boundaries of the domain.

4. Now, using the current values for ϕ^{HO} and J^{HO}, we need to compute the boundary conditions. In [18], computing the boundary condition requires (at the left endpoint) the values $J^{HO}_{\frac{1}{2}}$ and ϕ^{HO}_0. Using these moments, we can write an equation for ϕ^{LO}_0.

5. Given a new ϕ, $\hat{D}$, and boundary conditions, we can compute $F(\phi)$. This is done using the equation below:

$$F_i = -\frac{1}{3\Sigma_t}\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} + \frac{\hat{D}_{i+\frac{1}{2}}}{2\Delta x}(\phi_{i+1} + \phi_i) - \frac{\hat{D}_{i-\frac{1}{2}}}{2\Delta x}(\phi_i + \phi_{i-1}) + (\Sigma_t - \Sigma_s)\phi_i \ - Q_i \tag{2.56}$$

Using Newton-GMRES to solve $F(\phi) = 0$ provides a noticeable decrease in work. However, like with most iterative methods, preconditioning is extremely important. As we know, the goal of a preconditioner is to redistribute the spectrum of the linear operator in such a way as to please the iterative solver.

In [18], the suggested preconditioner is the inverse of the linear operator M:

$$M = \frac{\partial}{\partial x}\left[\frac{1}{3\Sigma_t}\frac{\partial}{\partial x} + \hat{D}^k\right] + (\Sigma_t - \Sigma_s) \tag{2.57}$$

where $\hat{D}^k$ denotes the value of $\hat{D}$ during the k^{th} nonlinear iteration.

At times, we may simplify this preconditioner slightly to make the implementation easier. In the simplified case, we use the inverse of $\bar{M}$:

$$\bar{M} = \frac{\partial}{\partial x}\left[\frac{1}{3\Sigma_t}\frac{\partial}{\partial x}\right] + (\Sigma_t - \Sigma_s) \tag{2.58}$$

While $\bar{M}$ does not approximate the inverse as well as M, it is still a remarkable improvement of the non-preconditioned problem. In general, we see that using the full preconditioner, M, reduces the number of function evaluations slightly over $\bar{M}$. In Chapter 6, we find that noisy evaluations of $\hat{D}$ may make the simplified preconditioner a better choice.

2.5.2 Quasi-Diffusion (QD)

Much like Nonlinear Diffusion Acceleration, or NDA, Quasi-Diffusion is a moment based accelerator [14, 2, 18]. With Quasi-Diffusion, we compute the zero-th and first moments of the transport equation. Recall from the previous section that the 0^{th} moment can be written as:

$$\frac{dJ}{dx} + (\Sigma_t - \Sigma_s)\phi = Q. \tag{2.59}$$

Now, let us compute the 1^{st} moment of the transport equation:

$$\begin{aligned}
\int_{-1}^{1}\left[\mu^2\frac{\partial}{\partial x}\psi(x,\mu)+\Sigma_t\mu\psi(x,\mu)\right]d\mu &= \int_{-1}^{1}\left[\frac{1}{2}\left(\Sigma_s\mu\phi(x)+\mu Q(x)\right)\right]d\mu \\
\int_{-1}^{1}\mu^2\frac{\partial}{\partial x}\psi(x,\mu)d\mu+\Sigma_t J &= 0 \\
\frac{\partial}{\partial x}\int_{-1}^{1}\mu^2\psi(x,\mu)d\mu+\Sigma_t J &= 0 \\
\frac{\partial}{\partial x}\int_{-1}^{1}\mu^2\psi(x,\mu)d\mu\left(\frac{\int_{-1}^{1}\psi(x,\mu)d\mu}{\int_{-1}^{1}\psi(x,\mu)d\mu}\right)+\Sigma_t J &= 0 \\
\frac{\partial E\phi}{\partial x}+\Sigma_t J &= 0
\end{aligned} \tag{2.60}$$

where

$$E(x)=\frac{\int_{-1}^{1}\mu^2\psi(x,\mu)d\mu}{\int_{-1}^{1}\psi(x,\mu)d\mu}. \tag{2.61}$$

$E(x)$ is known as the Eddington tensor [2, 18], or in one-dimension, the Eddington Factor.

In Equation 2.60 we solve for J to find

$$J=-\frac{1}{\Sigma_t}\frac{\partial E\phi}{\partial x}. \tag{2.62}$$

Now, we can substitute Equation 2.62 into Equation 2.41 to create our Low-Order Equation for Quasi-Diffusion. This yields

$$\frac{d}{dx}\left(-\frac{1}{\Sigma_t}\frac{\partial E\phi}{\partial x}\right)+(\Sigma_t-\Sigma_s)\phi=Q. \tag{2.63}$$

Prior to discretization, no approximations have been made and Equation 2.63 is exact. Upon discretization, Equation 2.63 becomes

$$\left(\frac{-1}{\Sigma_t}\right)\frac{E^{HO}_{i+1}\phi_{i+1}-2E^{HO}_{i}\phi_i+E^{HO}_{i-1}\phi_{i-1}}{\Delta x^2}+(\Sigma_t-\Sigma_s)\phi_i=Q_i. \tag{2.64}$$

In Equation 2.64, E^{HO} is computed using the high-order angular flux computed by executing a transport sweep using the previous scalar flux to build the scattering source.

Picard (Classic) Quasi-Diffusion

Picard, or Classic, Quasi-Diffusion works quite similarly to Picard NDA. We cycle back and forth between the high- and low-order problems until we have found a solution. The iteration process is as follows:

1. Begin with some initial guess ϕ_0. Set iteration counter $i = 0$.
2. Execute a transport sweep to recover $\psi_{i+\frac{1}{2}}$.
3. Use $\psi_{i+\frac{1}{2}}$ to compute $E_{i+\frac{1}{2}}$.
4. Solve Equation 2.63 for ϕ_{i+1} with $E = E_{i+\frac{1}{2}}$.
5. Check to see if $||\phi_{i+1} - \phi_i||_2 < tolerance$. If so, return declaring ϕ_{i+1} the solution. Otherwise, increment $i = i + 1$ and return to Step (2).

Again, much like NDA, Quasi-Diffusion provides a significant reduction in work. However, while in the continuous form, if ϕ^* is a solution to the high-order problem, it is guaranteed to be a solution to the low-order problem, without a correction term, QD lacks this consistency in the discrete case. This means that upon convergence, we may not have $\phi^{HO} = \phi^{LO}$. We can fix this by adding a correction term, γ, in Equation 2.63:

$$\frac{d}{dx}\left(-\frac{1}{\Sigma_t}\frac{\partial E\phi}{\partial x} + \gamma\phi\right) + (\Sigma_t - \Sigma_s)\phi = Q. \tag{2.65}$$

The consistency term (or correction term), γ is given by

$$J^{HO} = -\frac{1}{\Sigma_t}\frac{dE^{HO}\phi^{HO}}{dx} + \gamma\phi^{HO}. \tag{2.66}$$

Solving for γ and discretizing yields

$$\gamma_{i+1/2} = \frac{\frac{1}{\Sigma_t}\frac{E^{HO}\phi^{HO}_{i+1} - E^{HO}\phi^{HO}_i}{\Delta x} + J^{HO}_{i+1/2}}{\phi^{HO}_{i+1/2}}. \tag{2.67}$$

In essence, γ is used only to match the discretization errors in the high-order and low-order equations. As the mesh is refined, γ should approach zero everywhere in the domain. We can see that the formulas for γ and $\hat{D}$ from NDA look very similar, however they serve different purposes. While $\hat{D}$ makes the continuous form of the NDA low-order equation consistent with the high-order equation, the QD low-order problem is already consistent in its continuous form.

2.6 Diffusion Synthetic Acceleration

Diffusion Synthetic Acceleration (DSA) [2] can be thought of as a preconditioned source iteration. With source iteration, we took an approximation to the scalar flux, $\phi^{(n)}$, executed a single transport sweep to recover $\psi^{(n+1)}$, and then integrated the angular flux to recover the

new scalar flux, $\phi^{(n+1)}$. These physics-based preconditioned Picard iterations apply an update formula before computing $\phi^{(n+1)}$.

Diffusion Synthetic Acceleration works as follows:

1. Input $\phi^{(n)}$.
2. Execute a single transport sweep to recover $\psi^{(n+\frac{1}{2})}$.
3. Integrate over angle to recover $\phi^{(n+\frac{1}{2})}$.
4. Compute $\Delta\phi = \phi^{(n+\frac{1}{2})} - \phi^{(n)}$.
5. Solve the modified diffusion equation for F,

$$-\frac{d}{dx}\frac{1}{3\Sigma_t(x)}\frac{dF}{dx}(x) + \Sigma_a(x)F(x) = \Sigma_s\Delta\phi. \tag{2.68}$$

   subject to the boundary conditions

$$\begin{aligned} F(0) - \frac{2}{3\Sigma_t(0)}\frac{dF}{dx}(0) &= 0, \text{ in the case of a left vacuum boundary, and,} \\ \frac{dF}{dx}(\tau) &= 0, \text{ in the case of a right reflective boundary..} \end{aligned}$$

6. Apply the update formula $\phi^{(n+1)} = \phi^{(n+\frac{1}{2})} + F$

This process, through a series of steps, takes as input $\phi^{(n)}$ and returns a new scalar flux $\phi^{(n+1)}$. We can represent this as

$$\phi^{(n+1)} = G_{DSA}(\phi^{(n)}). \tag{2.69}$$

Much like source iteration, we have found the solution to the neutron transport equation when we've found ϕ such that

$$\phi = G_{DSA}(\phi). \tag{2.70}$$

It is expected that the DSA preconditioned source iteration will converge in significantly fewer iterations than source iteration. It is important to note that there is a small cost incurred by computing the DSA update, but in 1-D, it amounts to a single tri-diagonal matrix solve. This work is inconsequential compared to that of a transport sweep, so it is generally ignored.

Much like with source iteration or Picard-NDA, we can also apply Anderson Acceleration to the fixed-point problem $\phi = G_{DSA}(\phi)$. Furthermore, the fixed-point map G_{DSA} is an affine map. Through a little bit of reorganization, we can solve the DSA fixed-point problem using GMRES as well.

2.7 Results

In this section, we discuss the convergence results for a series of test problems. This set of test problems will attempt to isolate difficult problems where source iteration converges slowly. We will see that most of these algorithms will be a huge improvement over source iteration.

2.7.1 Test Problems

The tests in Table 2.1 have scattering ratios of $c = .9$, .99, and .999. As the scattering ratio increases, source iteration takes more iterations to converge. We consider tests with different domain lengths $\tau = 1$ and 10, or 10 and 100 mean-free-paths, respectively. Furthermore, we use different spatial meshes to demonstrate the robustness of the various algorithms.

Table 2.1: One-Dimensional, One-Group, Slab Geometry Test Problems

Test	1	2	3	4	5
Σ_s	9	9.9	9.9	9.9	9.99
Σ_t	10	10	10	10	10
τ	1	1	10	10	10
N_x	50	50	50	500	500

For each test problem, we will converge the solution to a tolerance of 10^{-8}. In all cases where Anderson acceleration is used to accelerate convergence, we will store 5 vectors in the history. For Newton's method, we will choose our forcing term $\eta = .01$ for each test and limit ourselves to 15 Krylov iterations per Newton, though this cap is never reached.

Table 2.2 displays the convergence results for the tests described in Table 2.1. We compare 11 methods, including source iteration to give us a baseline iteration count. In Table 2.2 the iteration counts *only account for the number of transport sweeps to convergence.* In many of the algorithms (NDA, DSA, QD) we require a low-order matrix solve at each iteration. The matrix solves involve a tri-diagonal system which can be handled very efficiently in MATLAB. This work is negligible compared to the work required for a transport sweep and therefore is ignored. For example, when using Picard-DSA to solve the transport equation in MATLAB, the time spent in the diffusion solve is merely .13% of the time spent computing a transport sweep when 500 spatial grid points and 20 angular grid points are used.

Looking at Table 2.2, we can see that many of these algorithms are very competitive. For all test problems, GMRES-DSA performs exceptionally well. Additionally, for test problems 1, 2, 4 and 5 Anderson Accelerated NDA/QD perform very well. Anderson Accelerated DSA

Table 2.2: One-Dimensional, One-Group, Slab Geometry Test Problems Convergence Results

Method	Test 1	Test 2	Test 3	Test 4	Test 5
Source Iteration	123	454	1579	1513	11869
GMRES	12	13	27	53	59
Picard NDA	11	12	DNC[1]	12	13
Picard QD	10	12	DNC[1]	12	13
JFNK-NDA	17	19	32	19	21
Picard - DSA	17	23	14	23	26
GMRES - DSA	7	7	7	7	7
Anderson Accelerated SI	20	32	222	212	1022
Anderson Accelerated NDA	8	9	33	9	10
Anderson Accelerated QD	8	9	30	9	10
Anderson Accelerated DSA	9	9	10	10	11

performs very well across the board as well. Furthermore, for test problems 1, 2, 4 and 5, Picard NDA and Picard QD require only a few extra transport sweeps.

Special attention should be paid to Test Problem 3. This problem was chosen since it is known [18] that Picard-NDA diverges for some problems in which cells become too thick. For this test, each cell is quite large in terms of mean-free-paths (2 m.f.p's per cell). Compare this to Test Problem 4 in which each cell is only one-fifth of a mean-free-path wide. It is clear that for Test Problem 3, the mesh is not fine enough to resolve the physics appropriately and thus Picard-NDA cannot converge.

2.8 Moving Forward

At this point, we have offered a very thorough treatment of the state-of-the-art methods for solving 1-D, single-material, single-group fixed source problems with isotropic scattering. This chapter has highlighted the idea of the transport sweep which will be used within each of the algorithms throughout the remainder of this thesis. The lessons we have learned from these simple problems will guide us when we move to multiple dimensions, multi-material, multi-group problems and when we begin solving the k-eigenvalue problem.

Table 2.2 will guide our focus when we begin looking at more difficult problems with the exception of Quasi-Diffusion. When we move to multi-dimensional hybrid problems where the transport sweep is executed using Monte Carlo simulation, we will ignore QD for a few basic reasons. The main reason we will ignore QD is because it requires us to resolve the Eddington tensor. Trying to tally a tensor using Monte Carlo simulation will be very tedious and extremely

[1]Did not converge due to spatial mesh size.

noisy. It will require far too many Monte Carlo realizations to get an accurate representation of the tensor. For this reason, we'll focus our attention towards NDA and DSA which do not require us to tally this noisy quantity.

Chapter 3

Solving the 2D Transport Equation

Much like we did in Chapter 2, let us consider several methods for solving the 2-D neutron transport equation. We'll begin by describing the 2-D transport equation and work to gain a functional understanding of the problem domain. After this, we'll discuss the transport sweep in two spatial dimensions. Once we have a method for executing a transport sweep, we'll discuss the implementation of source iteration. Finally, we'll revisit the acceleration techniques encountered in Chapter 2.

3.1 Introduction

In Chapter 1 we derived the full 3-D transport equation, Equation 1.12. For now, let us ignore the fission term and consider time-independent, fixed source problems. In this case, Equation 1.12 reduces to

$$\hat{\Omega} \cdot \nabla\psi(\vec{r}, E, \hat{\Omega}) + \Sigma_t(\vec{r}, E)\psi(\vec{r}, E, \hat{\Omega}) = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ \Sigma_s(\vec{r}, \hat{\Omega}' \to \hat{\Omega}, E' \to E)\psi(\vec{r}, E', \hat{\Omega}') + S_{ext}(\vec{r}, \hat{\Omega}, E) \quad (3.1)$$

We cannot solve the transport equation in its full continuous form, so we must discretize Equation 3.1 in space, energy and angle and we will follow the example set in [21]. First, we perform the energy discretization, yielding the system of multigroup equations. The multigroup equations were derived in Section 1.2.1. Thus, for $g = 1, ..., G$, we have:

$$\hat{\Omega} \cdot \nabla\psi^g(\vec{r}, \hat{\Omega}) + \Sigma_t^g(\vec{r})\psi(\vec{r}, \hat{\Omega}) = \sum_{g'=1}^{G} \int_{4\pi} d\hat{\Omega}' \ \Sigma_s^{g' \to g}(\vec{r}, \hat{\Omega} \cdot \hat{\Omega}')\psi^{g'}(\vec{r}, \hat{\Omega}') + S_{ext}^g(\vec{r}, \hat{\Omega}) \quad (3.2)$$

In order to represent this equation more succinctly, we'll represent the right hand side as a

single function, $Q(\vec{r}, \hat{\Omega})$:

$$\hat{\Omega} \cdot \nabla \psi^g(\vec{r}, \hat{\Omega}) + \Sigma_t^g(\vec{r}) \psi(\vec{r}, \hat{\Omega}) = Q^g(\vec{r}, \hat{\Omega})$$

where

$$Q^g(\vec{r}, \hat{\Omega}) = \sum_{g'=1}^{G} \int_{4\pi} d\hat{\Omega}' \ \Sigma_s^{g' \to g}(\vec{r}, \hat{\Omega} \cdot \hat{\Omega}') \psi^{g'}(\vec{r}, \hat{\Omega}') + S_{ext}^g(\vec{r}, \hat{\Omega}).$$

In Section 1.2.2, we discussed the angular discretization of the transport equation. This leaves us with a system of equations

$$\hat{\Omega}_n \cdot \nabla \psi_n^g(\vec{r}) + \Sigma_t^g(\vec{r}) \psi_n(\vec{r}) = Q_n^g(\vec{r}) \text{ for } g = 1, ..., G, \text{ and } n = 1, ..., 4N \tag{3.3}$$

with the following definitions:

$$\begin{aligned} \psi_n^g(\vec{r}) &= \psi^g(\vec{r}, \hat{\Omega}_n) \\ Q_n^g(\vec{r}) &= Q^g(\vec{r}, \hat{\Omega}_n). \end{aligned}$$

After discretizing in energy and angle, we are left with the task of discretizing in space. We'll devote the next subsection to accomplishing this task.

3.1.1 Spatial Domain and Spatial Discretization

We will only consider rectangular domains in this section and discretize using rectangular grid cells. Suppose we consider I spatial cells in the x-direction and J spatial cells in the y-direction. This results in an $x - y$ domain as seen in Figure 3.1.

Here, we notice that cell centers are indexed by integers and cell edges are indexed by half integers. We choose to discretize the domain such that each cell is comprised of a single material. That is, the total, scattering, absorption and fission cross-sections are constant within a given cell, and thus can be represented as $\Sigma_{i,j}$. This may mean that we need to approximate cross-section for cells in which multiple materials would be present, for example along a curved material interface.

For each cell and each angle, we'll define a cell-averaged flux

$$\psi_{n,i,j} = \frac{1}{\Delta x_i \Delta y_j} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \psi_n(x, y) \ dy \ dx \tag{3.4}$$

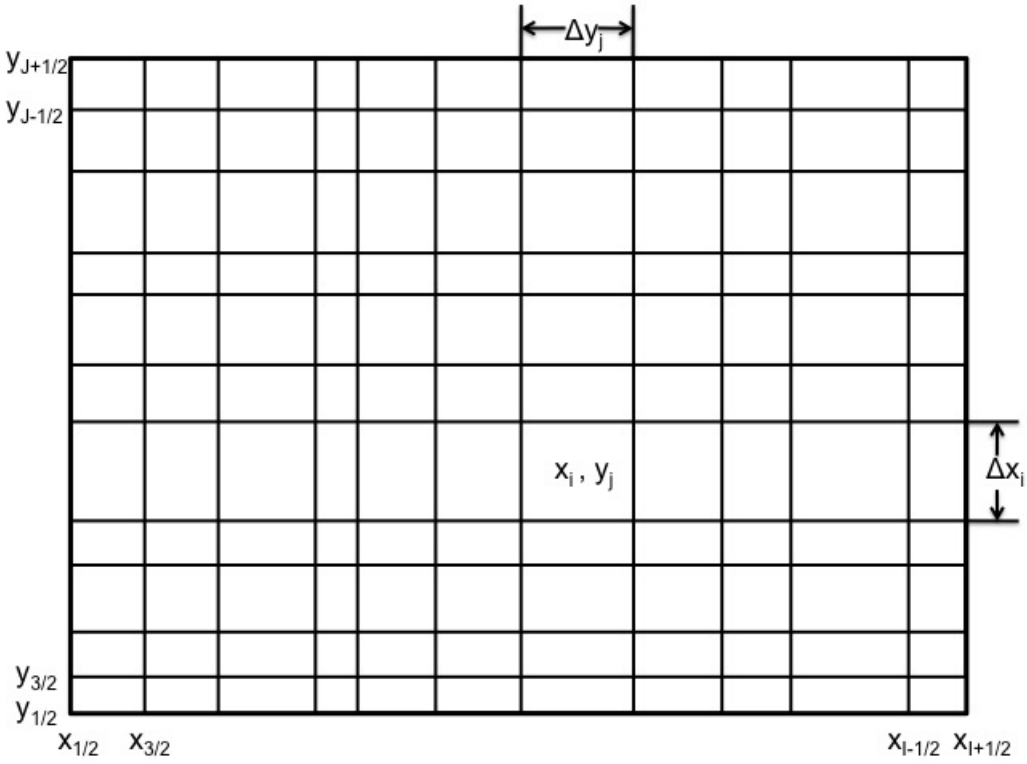

Figure 3.1: $x - y$ domain for 2D transport

and cell-edge fluxes

$$\psi_{n,i+\frac{1}{2},j} = \frac{1}{\Delta y_j} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \psi_n(x_{i+\frac{1}{2}}, y)\ dy \tag{3.5}$$

$$\psi_{n,i,j+\frac{1}{2}} = \frac{1}{\Delta x_j} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \psi_n(x, y_{j+\frac{1}{2}})\ dx. \tag{3.6}$$

Now, integrating Equation 3.3 over a single grid-cell and dividing by $\Delta x_i \Delta y_j$ yields

$$\frac{1}{\Delta x_i \Delta y_j} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \left(\hat{\Omega}_n \cdot \nabla \psi_n^g(\vec{r}) + \Sigma_t^g(\vec{r}) \psi_n(\vec{r}) \right)\ dy\ dx$$
$$= \frac{1}{\Delta x_i \Delta y_j} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} Q_n(\vec{r})\ dy\ dx.$$

After some algebra, we find that

$$\frac{\mu_n}{\Delta x_i} \left(\psi_{n,i+\frac{1}{2},j} - \psi_{n,i-\frac{1}{2},j} \right) + \frac{\eta_n}{\Delta y_j} \left(\psi_{n,i,j+\frac{1}{2}} - \psi_{n,i,j-\frac{1}{2}} \right) + \Sigma_{t,i,j} \psi_{n,i,j} \quad = \quad q_{n,i,j}$$

where

$$q_{n,i,j} = \frac{1}{\Delta x_i \Delta y_j} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} Q_n(x, y)\ dy\ dx. \tag{3.7}$$

As is noted in [21], no approximations have been made in the spatial discretization at this point in time. However, as it is written, Equation 3.7 has too many unknowns. We need a way to relate the cell-averaged angular flux to the cell-edge fluxes. We do this using the diamond difference approximation:

$$\psi_{n,i,j} = \frac{1}{2} \left(\psi_{n,i+\frac{1}{2},j} + \psi_{n,i-\frac{1}{2},j} \right) \tag{3.8}$$

$$\psi_{n,i,j} = \frac{1}{2} \left(\psi_{n,i,j+\frac{1}{2}} + \psi_{n,i,j-\frac{1}{2}} \right) \tag{3.9}$$

Now, let us assume that for a given cell (i, j) and $\hat{\Omega}_n$ such that $\mu_n > 0$ and $\eta_n > 0$ the cell-edge fluxes on the left and bottom boundaries are known. Now, we have three equations (Equations 3.7, 3.8, 3.9) and three unknowns ($\psi_{n,i,j}$, $\psi_{n,i+\frac{1}{2},j}$, $\psi_{n,i,j+\frac{1}{2}}$). This linear system is trivial to solve; we can compute the cell-averaged flux by utilizing the following equation

$$\psi_{n,i,j} = \left[\Sigma_{t,i,j} + \frac{2\mu_n}{\Delta x_i} + \frac{2\eta_n}{\Delta y_j} \right]^{-1} \left[q_{n,i,j} + \frac{2\mu_n}{\Delta x_i} \psi_{n,i-\frac{1}{2},j} + \frac{2\eta_n}{\Delta y_j} \psi_{n,i,j-\frac{1}{2}} \right] \tag{3.10}$$

and substituting this value into the diamond difference relations we can obtain the unknown

cell-edge fluxes. In fact, as long as one of the vertical cell-edge fluxes and one of the horizontal cell-edge fluxes is known, we can solve Equation 3.7 for a given cell. We'll use this idea when completing the 2D transport sweep.

3.1.2 2D Transport Sweep

Much like we did in one spatial dimension, we'll sweep the mesh one angle at a time. Let us choose one of the discrete angles $\hat{\Omega}_n$ such that $\mu_n > 0$ and $\eta_n > 0$ and, for now, let us assume that all of the boundary conditions are explicit (not dependent on the flux near the boundary). Then, the cell-edge incoming fluxes are known for all exterior edges where either $x = x_{\frac{1}{2}}$ or $y = y_{\frac{1}{2}}$, as shown in Figure 3.2.

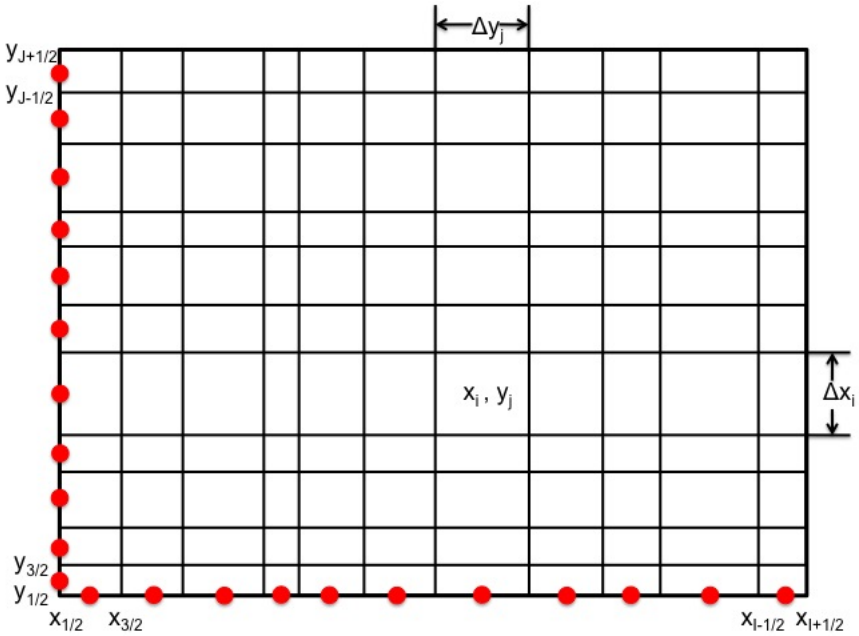

Figure 3.2: For $\mu_n > 0$, $\eta_n > 0$, the cell-edge incoming angular fluxes denoted by red-circles are known since the boundary conditions are explicit.

Now, let us consider cell $(1, 1)$. As we can see in Figure 3.3, two cell-edge fluxes are known for cell $(1, 1)$, so we can compute the cell-averaged flux $\psi_{n,1,1}$ using Equation 3.10.

Then, using the diamond difference relations, we can compute the cell-edge fluxes on the top and right boundaries of cell $(1, 1)$, as shown in Figure 3.4.

Moving along, we can now sweep right across the first row of our grid, computing the

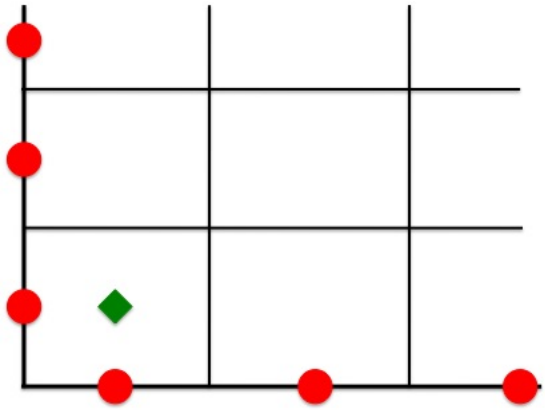

Figure 3.3: For $\mu_n > 0$, $\eta_n > 0$, the cell-edge fluxes denoted by red-circles are known since the boundary conditions are explicit. The cell-centered value, denoted by the green diamond, can be computed using Equation 3.10.

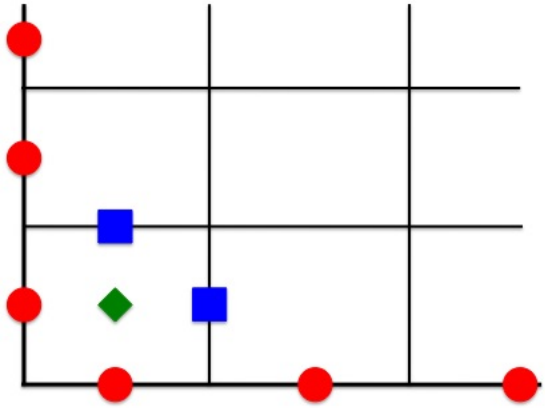

Figure 3.4: Given the cell-edge fluxes on the left and bottom boundaries along with the cell-averaged flux, we can compute the cell-edge fluxes, denoted by blue squares, on the top and right boundaries using Equations 3.8 and 3.9.

cell-averaged fluxes and the unknown cell-edge fluxes, as demonstrated in Figure 3.5.

After the first row is complete, we can repeat this process in the second row. Since the first row has been completed, cell $(1, 2)$ now has a known flux on the left and bottom boundaries, allowing us to compute the cell-averaged flux for this cell. This is shown in Figure 3.6.

We continue this process of beginning in the first cell in a given row and sweep to the right across the grid until we have computed all of the unknown cell-averaged and cell-edge fluxes

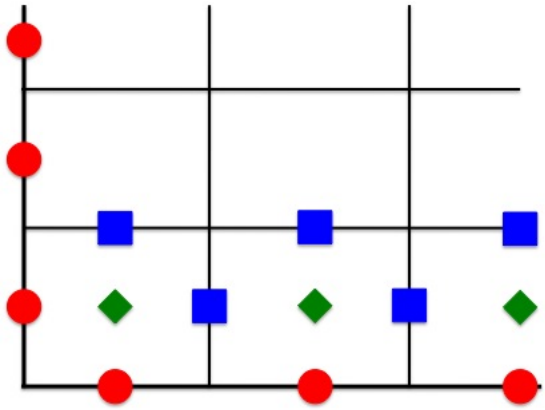

Figure 3.5: We proceed to the right across the first row computing the unknown cell-averaged and cell-edge fluxes.

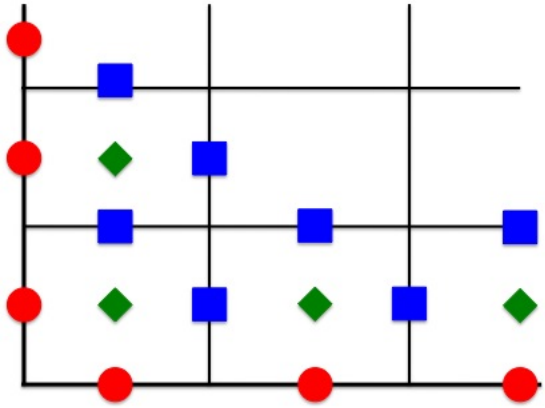

Figure 3.6: Evaluating unknown fluxes in the second row of our grid.

in that row. Once that row is complete, we move on to the next row, until we have swept to the top of the grid. At this point, we repeat this entire process for the next angle in the first quadrant.

We step through angles in Quadrants II, III and IV in a similar manner. For example, for angles in Quadrant III ($\mu < 0$, $\eta < 0$), the boundary conditions are known for cell-edges lying on the top of the grid and the right side of the grid. For these angles, we start in the top right corner of the grid and sweep left, then down, similar to the way we swept through the grid for angles in Quadrant I.

3.1.3 Dealing with Reflective Boundary Conditions

In the previous section, we dealt only with rectangular domains in which all of the boundary conditions were explicit (either a vacuum or given by a function which does not depend on either the angular or scalar flux near the boundary). However, reflective boundary conditions play a major role in reactor modeling and require some advanced treatment.

Left Boundary Reflective

We'll begin by looking at a single reflective boundary condition, and let us choose for it to be the left boundary as shown in Figure 3.7. Now, let's pick a pair of angles $\hat{\Omega}_1 = (\mu, \eta)$ and $\hat{\Omega}_2 = (-\mu, \eta)$ where both $\mu > 0$ and $\eta > 0$. Then for a point $\vec{r}_0$ on the left boundary, $\psi(\vec{r}_0, \hat{\Omega}_1) = \psi(\vec{r}_0, \hat{\Omega}_2)$, as shown in Figure 3.8.

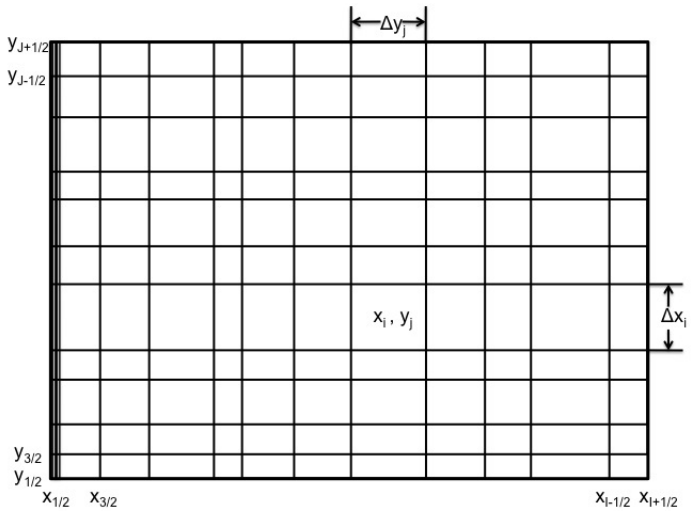

Figure 3.7: Spatial domain with three explicit boundaries (single black line) and one reflective boundary (triple black line).

After considering Figures 3.7 and 3.8, it should be clear that we must know the value of $\psi(\vec{r}_0, \hat{\Omega}_2)$ before we can begin a sweep across the domain for an angle $\hat{\Omega}_1$. In this case, we sweep the domain in the following manner:

1. Choose an angle $\hat{\Omega} = (-\mu, \eta)$ where $\mu, \eta > 0$.
2. Beginning from the first row, right-most cell of the domain, sweep left across the domain until we encounter the left boundary.

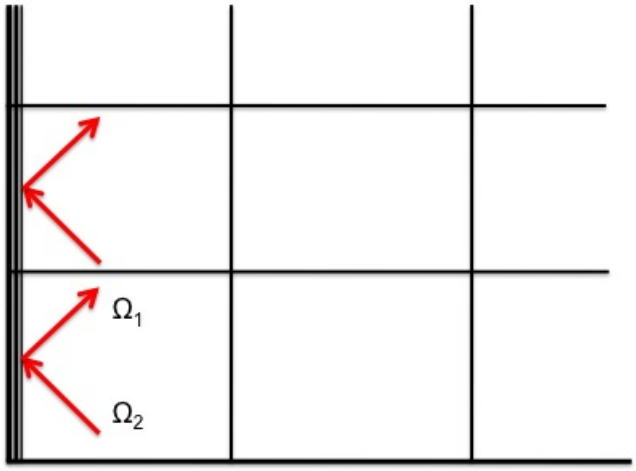

Figure 3.8: Particles encountering the reflective boundary with angle $\hat{\Omega}_2$ are reflected off in a new direction $\hat{\Omega}_1$.

3. Set $\psi(\vec{r}_0, \hat{\Omega}) = \psi(\vec{r}_0, \hat{\Omega}^*)$ where $\hat{\Omega}^* = (\mu, \eta)$.

4. Sweep right across the domain until we encounter the right boundary.

5. Repeat steps 2 - 4, moving up one row at a time, until we've swept the entire domain.

This sweep method is depicted in Figure 3.9. We use this scheme for all pairs of angles with $\eta > 0$. When $\eta < 0$, we use a similar scheme for sweeping the domain, but instead of starting in the lower-right corner of the domain, we begin from the upper right corner of the domain. One should notice that this sweep method is no more expensive than a sweep with all four boundaries explicit. The same number of passes left and right across the domain take place, only the order in which they're performed has changed. This is not the case in the next section when we consider reflective boundaries on the left and right.

Left and Right Boundary Reflective

In the previous case, with only one reflective boundary, we were easily able to determine incoming fluxes on the left boundary by simply executing a sweep from right to left in the reflected direction $\hat{\Omega}^*$. By utilizing the known incoming flux on the right boundary, we can propagate flux information across the domain to the left boundary and now have all the information required to sweep back across the domain. This becomes far more complicated when a pair of opposing boundaries are both reflective. In this case, the incoming flux on the right boundary is dependent on the incoming flux on the left boundary and vice versa. With this problem con-

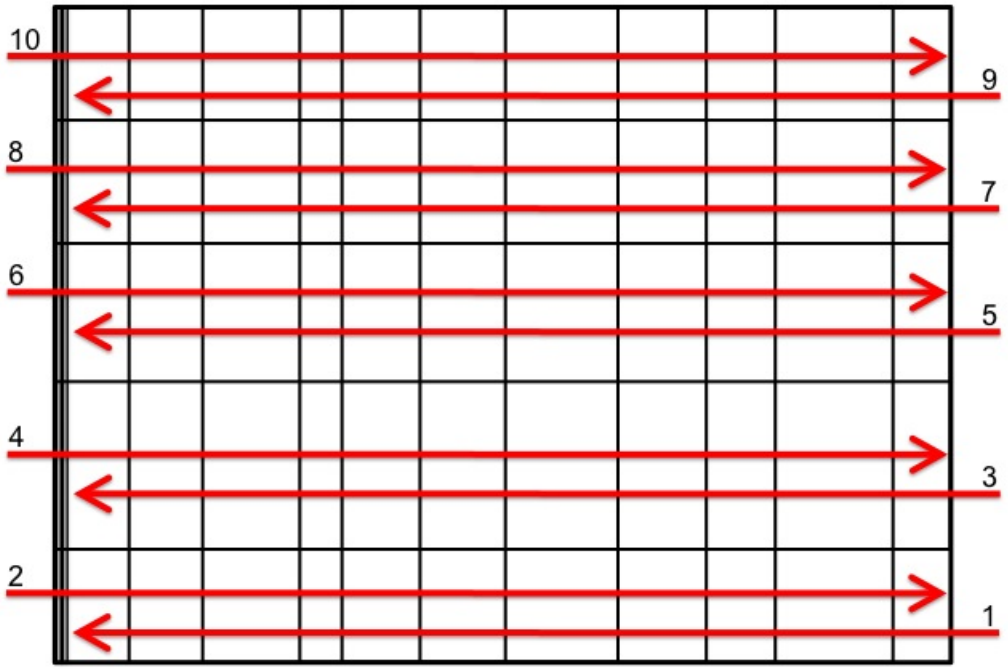

Figure 3.9: Particles encountering the reflective boundary with angle $\hat{\Omega}_2$ are reflected off in a new direction $\hat{\Omega}_1$.

figuration, we are forced to iterate on the boundary conditions in order to determine incoming fluxes on each side.

In this case, we sweep the domain according to the following plan:

1. Choose an angle $\hat{\Omega} = (-\mu, \eta)$ where $\mu, \eta > 0$.
2. Beginning from the first row, right-most cell of the domain, make an initial guess for $\psi(\vec{r}_{right}, \hat{\Omega}) = \Psi^{old}_{right}$.
3. Sweep left across the domain until we encounter the left boundary.
4. Set $\psi(\vec{r}_0, \hat{\Omega}) = \psi(\vec{r}_{left}, \hat{\Omega}^*)$ where $\hat{\Omega}^* = (\mu, \eta)$.
5. Sweep right across the domain until we encounter the right boundary.
6. Let $\Psi^{new}_{right} = \psi(\vec{r}_{right}, \hat{\Omega}^*)$.
7. If $\|\Psi^{new}_{right} - \Psi^{old}_{right}\| < \tau$ where τ is a user-defined convergence tolerance, move on to Step 8, otherwise set $\Psi^{old}_{right} = \Psi^{new}_{right}$ and repeat Steps 2 - 6.
8. Move up to the next row and repeat Steps 2 - 7 until we've swept the entire domain.

This scheme can be visualized in Figure 3.10.

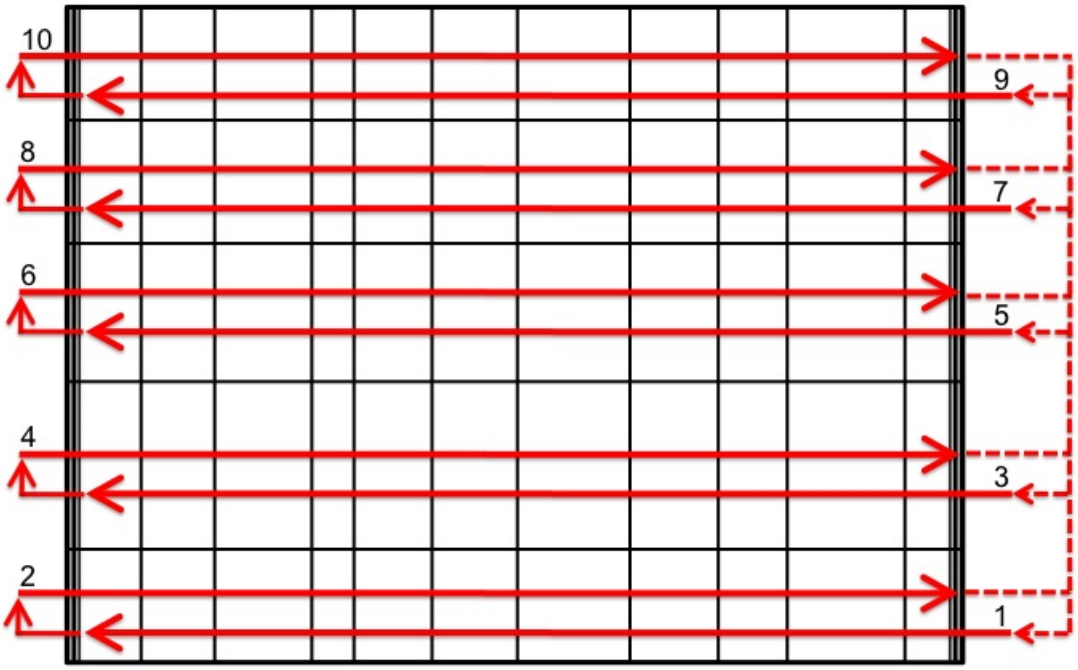

Figure 3.10: We must iterate on the incoming flux until we've reached an acceptable level of convergence when there is a pair of opposing reflective boundaries.

Again, we use this scheme for all pairs of angles $\hat{\Omega}$ and $\hat{\Omega}^*$ with $\eta > 0$. When $\eta < 0$, we start at the top of the grid and work our way down using a similar scheme. It is important to note that the mesh sweep for this configuration can be several times more expensive than a mesh sweep when all boundaries are explicit. For optically thick problems, in practice we see that 2 - 4 iterations are needed for the boundary conditions to converge, depending on the accuracy of our initial guess. For smaller domains where the flux from the boundary has a greater impact on the entire grid, more iterations may be required. If all boundary conditions are guessed exactly correct, this scheme will be exactly as expensive as the case of all explicit boundary conditions, but in general we should expect a sweep for this configuration to be at least twice as expensive.

Left and Bottom Boundary reflective

The final case which we will consider at this point is the configuration in which we have reflective boundaries on a single corner and explicit boundary conditions elsewhere. For simplicity, let us only consider the lower-left corner to be the corner with reflective boundary conditions, as shown in Figure 3.11.

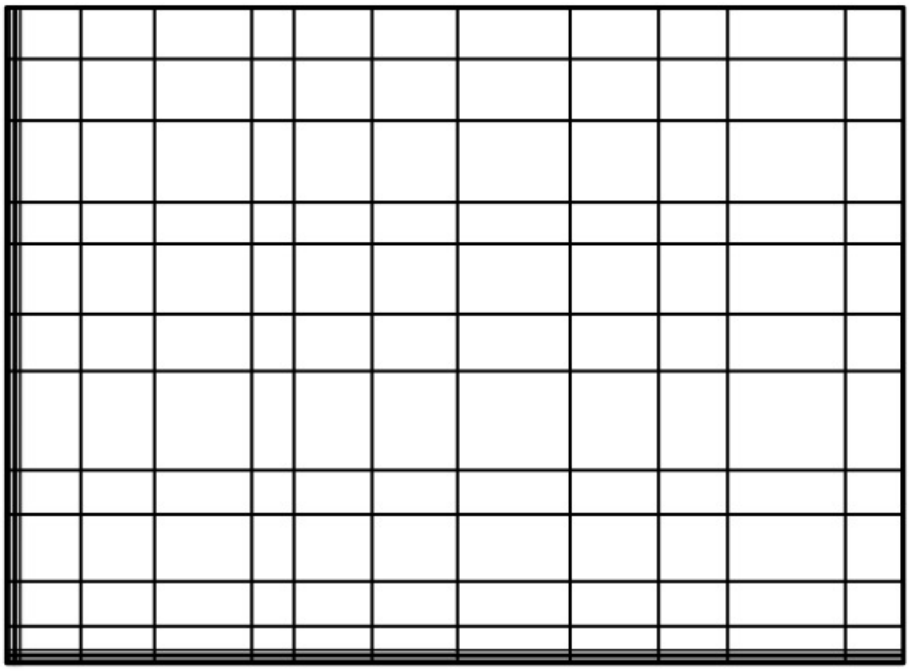

Figure 3.11: In this configuration the left and bottom boundaries are reflective and the remaining boundaries have explicit boundary conditions.

It should be clear that the incoming fluxes are known for both the top and right boundaries. Since these incoming fluxes are known, we can execute a mesh sweep for all angles $\hat{\Omega} = (-\mu, -\eta)$ with $\mu, \eta > 0$. For these angles we'll sweep from right to left, top to bottom.

After the sweep along $\hat{\Omega}$ has been completed, we now have partial information for the incoming fluxes on the left and bottom boundaries. Now, on the bottom boundary, we know the incoming flux for all directions $\hat{\Omega}^*$ such that $\mu < 0$ and $\eta > 0$. Similarly, on the left boundary, we know the incoming fluxes for all directions $\hat{\Omega}^{**}$ with $\mu > 0$ and $\eta < 0$. With this information, we can sweep all angles $\hat{\Omega}^*$ starting from the lower-right corner and all angles $\hat{\Omega}^{**}$ starting from the upper-left corner.

Finally, after all angles $\hat{\Omega}$, $\hat{\Omega}^*$, and $\hat{\Omega}^{**}$ are completed, we have complete knowledge of the

incoming fluxes on the left and bottom boundaries. Therefore, we can finish the mesh sweep by sweeping all angles $\hat{\Omega}^{***}$ where $\mu, \eta > 0$ starting from the lower-left corner.

It is important to note that while we've had to change the order in which the mesh is swept, the sweep for configurations with a reflective corner is no more expensive than a sweep for a configuration with all explicit boundary conditions. While we do not initially have information regarding incoming fluxes on the left and bottom faces of the domain, we can obtain this information without iterating. For this reason, the sweep for a configuration with a reflective corner is most similar to the case where one reflective boundary exists.

3.1.4 Dealing with Negative Fluxes

It is important to note that for a given cell, even if the incoming fluxes are both positive, it is possible for the outgoing fluxes to become negative. Suppose that both $\psi_{i-\frac{1}{2},j}$ and $\psi_{i,j-\frac{1}{2}}$ are both known and positive. Then we can compute the cell-averaged flux using Equation 3.10. Then, by rearranging the diamond difference relations, we can compute $\psi_{i+\frac{1}{2},j}$ and $\psi_{i,j+\frac{1}{2}}$ by

$$\psi_{n,i+\frac{1}{2},j} = 2\left(\psi_{n,i,j} - \frac{1}{2}\psi_{n,i-\frac{1}{2},j}\right) \tag{3.11}$$

$$\psi_{n,i,j+\frac{1}{2}} = 2\left(\psi_{n,i,j} - \frac{1}{2}\psi_{n,i,j-\frac{1}{2}}\right) \tag{3.12}$$

This tells us that if either $\psi_{n,i,j} < \frac{1}{2}\psi_{n,i-\frac{1}{2},j}$ or $\psi_{n,i,j} < \frac{1}{2}\psi_{n,i,j-\frac{1}{2}}$, at least one of the outgoing fluxes will be negative. If Σ_t is large compared to Δx and Δy, there is a good possibility of encountering negative fluxes for cells where $q_{i,j}$ (the source) is small [21]. It is important to note that as we make the angular mesh more dense, we will need to refine the spatial mesh as well. We can see this by taking a closer look at Equation 3.10.

Negative fluxes do not reflect reality (they correspond to a negative distribution of neutrons) and should be avoided. There are two simple ways to alleviate this issue. First, we can use a finer mesh so that Δx and Δy are smaller in comparison to Σ_t. This is not necessarily an ideal solution, however. A finer mesh means more computation. If we double the number of cells in both the $x-$ and $y-$directions, we essentially quadruple the size of the problem. This increases the cost, in terms of memory and time, of solving this problem and this cost may become unreasonable before we've decreased cell sizes enough to avoid negative fluxes.

A second way to avoid negative fluxes is to set them equal to zero whenever they're encountered. That is, after the out-going fluxes are computed for a given cell, we check each of the outgoing fluxes and if either of them are negative, we set offending fluxes equal to zero.

A third way to avoid negative fluxes does exist, but does not use the diamond difference relation for computing outgoing fluxes. Instead, we solve for the exact angular flux along

characteristics by assuming a constant source inside the cell. This is referred to as the step characteristics method [21, 19] and will be discussed briefly in the next section.

Depending on our method for solving the problem, refining the mesh is potentially a better option if it is feasible. It may also be acceptable to allow negative fluxes for a few iterations. It can often be the case that as the scattering source gets larger we will eventually avoid these negative fluxes in later iterations. It is sometimes the case that while the angular flux may be negative at a location (x, y) for some direction $\hat{\Omega}_n$, it may be positive for all other directions. In general, this problem should be addressed on a case by case basis.

3.1.5 Step Characteristics Discretization

In this section we will briefly derive the step characteristics method as found in [19]. We find that this discretization is useful for large 2-D problems in which we cannot afford to refine the mesh.

We assume that neutrons travel in straight lines between collisions and that these straight lines are characteristics of transport equation, which are given by

$$\frac{d\psi}{ds} + \Sigma_t \psi = Q. \tag{3.13}$$

We will assume that within a given cell, Q and Σ_t are constant. We can integrate Equation 3.13 to recover

$$\psi = \psi_0 e^{-\Sigma_t s} + \int_0^s Q e^{-\Sigma_t (s-t)} dt \tag{3.14}$$

in which ψ_0 is the angular flux on the boundary and s is the distance along the characteristic to a point on the boundary.

Without loss of generality, let us assume that we are considering an angle in the first quadrant ($\mu > 0$ and $\eta > 0$). Furthermore, let us assume that the angular flux on the left and bottom boundaries of the cell do not vary in the y and x-directions, respectively. That is, we assume the angular flux along horizontal and vertical lines (cell-boundaries) in space can be written as step functions which depend on angle.

Now, let us draw a characteristic originating from the lower left corner of the cell. We can visualize this cell in Figure 3.12. Furthermore, let us translate the domain so the bottom left corner of the cell is coincident with the origin. Now, the angular flux at any point above or below the characteristic line can be written as

$$\psi_{below}(x, y, \mu, \eta) = Q(1 - e^{-\Sigma_t y/\eta})/\Sigma_t + \psi_{bottom} e^{-\Sigma_t y/\eta} \tag{3.15}$$

$$\psi_{above}(x, y, \mu, \eta) = Q(1 - e^{-\Sigma_t x/\mu})/\Sigma_t + \psi_{left} e^{-\Sigma_t x/\mu}. \tag{3.16}$$

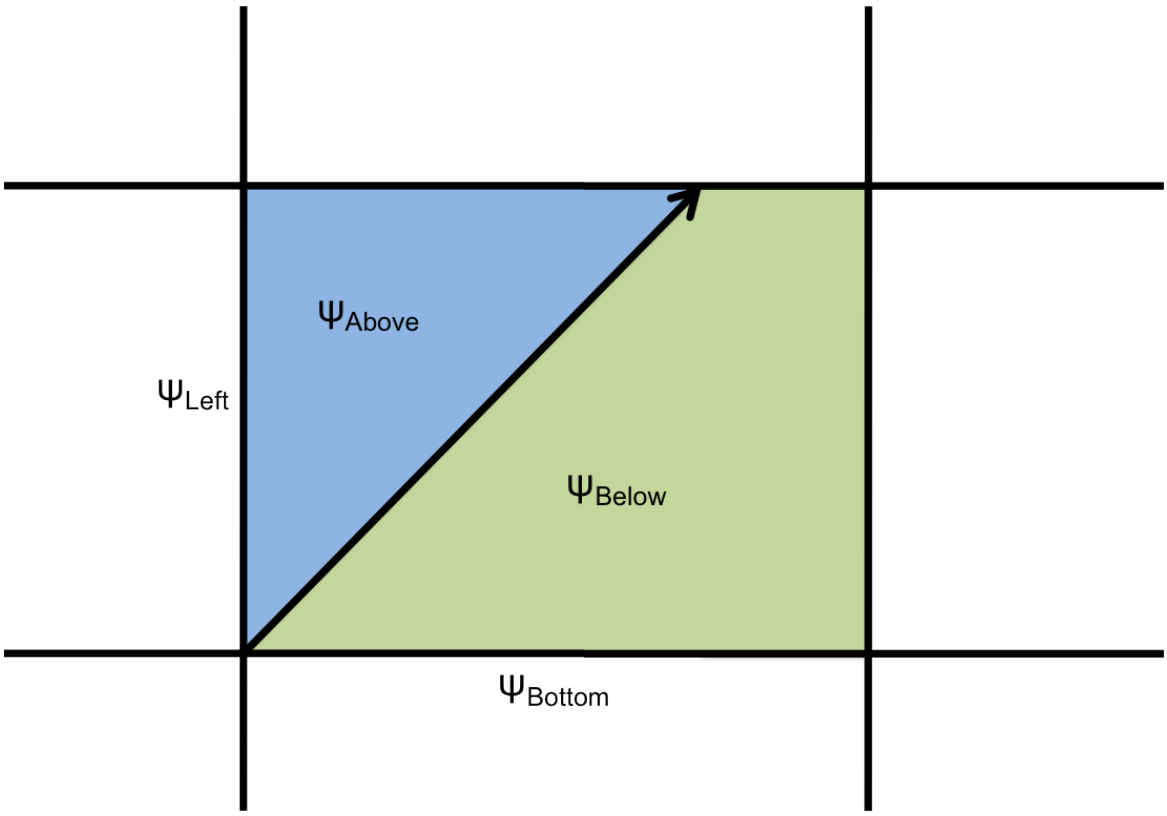

Figure 3.12: Configuration of cell for development of the step characteristics method.

Then, we can compute the average angular flux on the top and right boundaries using the following formulas

$$\psi_{right}(\Delta x, y, \mu, \eta) = \frac{1}{\Delta y}\int_0^{\Delta y} \psi(\Delta x, y, \mu, \eta)dy \tag{3.17}$$

$$\psi_{top}(x, \Delta y, \mu, \eta) = \frac{1}{\Delta x}\int_0^{\Delta x} \psi(x, \Delta y, \mu, \eta)dx. \tag{3.18}$$

It turns out that we can write an explicit formula for the average angular flux along the top and right boundaries. We have

$$\psi_{right} = S + (\psi_{left} - S)(1-\rho)e^{-\alpha} + (\psi_{bottom} - S)\rho(1-e^{-\alpha})/\alpha \tag{3.19}$$

$$\psi_{top} = S + (\psi_{left} - S)(1-e^{-\alpha})/\alpha \tag{3.20}$$

if $\rho < 1$ and

$$\begin{aligned} \psi_{right} &= S + (\psi_{bottom} - S)(1 - e^{-\beta})/\beta \quad (3.21) \\ \psi_{top} &= S + (\psi_{left} - S)(1 - e^{-\beta})/(\rho\beta) + (\psi_{bottom} - S)(1 - 1/\rho)e^{-\beta} \quad (3.22) \end{aligned}$$

if $\rho > 1$, where S, α, β and ρ are given by

$$\begin{aligned} S &= Q/\Sigma_t \\ \alpha &= \Sigma_t \Delta x/\mu \\ \beta &= \Sigma_t \Delta y/\eta \\ \rho &= \alpha/\beta. \end{aligned}$$

The value of ρ determines whether the characteristic passes through the top of the cell or the right side boundary. Now, the cell average angular flux can be recovered by a rearrangement of the relation

$$\mu \frac{\psi_{right} - \psi_{left}}{\Delta x} + \eta \frac{\psi_{top} - \psi_{bottom}}{\Delta y} + \Sigma_t \psi_{ave} = S. \quad (3.23)$$

This discretization guarantees that angular flux remains positive at all points in the domain. Unfortunately, this discretization is not second order accurate like the diamond difference discretization [19]. However, we will employ the step characteristics method for the 2-D eigenvalue problems considered in Chapter 4.

3.2 Source Iteration

As in one dimension, the most straightforward method for solving the 2-D transport equation is *source iteration*. Given a source term $Q(\vec{r}, \hat{\Omega})$, we can solve the equation

$$\hat{\Omega} \cdot \nabla \psi(\vec{r}, \hat{\Omega}) + \Sigma_t(\vec{r}) \psi(\vec{r}, \hat{\Omega}) = Q(\vec{r}, \hat{\Omega})$$

using the transport sweep described in Section 3.1.2. However, unless we are working in a non-scattering and non-fissioning environment, we do not know the total source term, Q, exactly. Recall, for a one-speed problem, Q is given by

$$Q(\vec{r}, \hat{\Omega}) = \int_{4\pi} d\hat{\Omega}' \; \Sigma_s(\vec{r}, \hat{\Omega} \cdot \hat{\Omega}') \psi(\vec{r}, \hat{\Omega}') + S_{ext}(\vec{r}, \hat{\Omega})$$

so in order to compute Q, we must already know the angular flux.

As in 1-D, we solve this problem iteratively. We take an initial guess, $\psi^{(0)}$, and use the

following relation to produce new iterates:

$$Q^{(n+1)}(\vec{r},\hat{\Omega}) = \int_{4\pi} d\hat{\Omega}'\ \Sigma_s(\vec{r},\hat{\Omega}\cdot\hat{\Omega}')\psi^{(n)}(\vec{r},\hat{\Omega}') + S_{ext}(\vec{r},\hat{\Omega}) \tag{3.24}$$

$$\hat{\Omega}\cdot\nabla\psi^{(n+1)}(\vec{r},\hat{\Omega}) + \Sigma_t(\vec{r})\psi^{(n+1)}(\vec{r},\hat{\Omega}) = Q^{(n+1)}(\vec{r},\hat{\Omega}) \tag{3.25}$$

The differences between source iteration in one and two dimensions are minute. Generally, since we cannot deal with the angular variable continuously, we represent that scattering term in terms of Legendre polynomials. In this case, instead of storing the angular flux, we choose to store several angular moments of the angular flux. These can be computed on the fly during a transport sweep. Fortunately, this means that the angular flux never needs to be stored. In 2-D, the group angular flux is a 4-D array whereas the group scalar flux and higher moments of the angular flux are only 2-D arrays.

In some cases, which we will demonstrate, source iteration may take hundreds or thousands of iterations to converge. As the spatial domain gets bigger, more energy groups are used and the angular grid is refined, this becomes an extremely expensive problem to solve. In this case, source iteration is not a reasonable method for solving the transport equation. Instead, more sophisticated methods like linear iterative solvers (e.g. Krylov methods), Diffusion Synthetic Acceleration or moment systems are used to accelerate convergence.

3.2.1 Multigroup Transport Sweep

At each iteration, we must execute a single transport sweep which we will represent

$$\Psi^{(n+1)} = \frac{1}{4\pi}\mathcal{L}^{-1}\left[(\mathcal{S}_U + \mathcal{S} + \mathcal{S}_L)\Phi^{(n)} + \vec{Q}\right], \tag{3.26}$$

in which $\mathcal{L}$ represents the streaming and absorption terms, $\mathcal{S}$ represents the in-group scattering, $\mathcal{S}_L$ represents the down-scattering matrix, and $\mathcal{S}_U$ represents the up-scattering contribution. However, in reality, we usually only execute the transport sweep one group at a time. For the first group, we build the scattering source using the group scalar fluxes entirely from iteration (n). However, for group 2, we can build the scattering source using the first group scalar flux we just computed at iteration level iteration $(n+1)$ and the rest of the scattering source from group fluxes at iteration level (n). In this sense, we should write

$$\Psi^{(n+1)} = \frac{1}{4\pi}\mathcal{L}^{-1}\left[(\mathcal{S}_U + \mathcal{S})\Phi^{(n)} + \mathcal{S}_L\Phi^{(n+1)} + \vec{Q}\right] \tag{3.27}$$

to indicate that we are using the most current group scalar fluxes when computing the scattering source for each group. In general, we represent this transport sweep using either Equation 3.26 or Equation 3.27, but we keep in mind the order in which this sweep is executed in practice.

3.3 Linear Iterative Methods

Just like we did in Chapter 2, we can represent the transport equation as an integral equation in terms of the scalar flux ϕ. In this sense, the transport equation can be again written as

$$\phi = \mathcal{K}\phi + g.$$

$\mathcal{K}$ and g now reflect the 2D nature of the problem, but we can define these operators in the same way we did in one spatial dimension. If we let $G(\phi)$ represent the action of a two-dimensional mesh sweep, we can write the operator $\mathcal{K}$ and g in terms of G. We have

$$g = G(\vec{0}) \tag{3.28}$$

$$\mathcal{K}\phi = G(\phi) - G(\vec{0}). \tag{3.29}$$

Therefore, solving the two-dimensional transport equation is as simple as solving the linear system,

$$M\phi = g,$$

where

$$M\phi = \phi - G(\phi) + G(\vec{0}).$$

We'll notice this is no different than one-dimension, except that ϕ has dimension $N_x \times N_y$ where N_i represents the number of cells in the i-direction.

Applying Krylov methods to this system of linear equations is no more difficult than it was in one-dimension, aside from memory concerns. Just as in Chapter 2, we can try to solve this system with GMRES, TFQMR or BiCGSTAB. However, with the increased dimensionality, we may not be able to store the entire Krylov basis built by GMRES in memory. In this case, we'll remind the reader that we can attempt to use GMRES with restarts, low-storage Krylov methods, or turn to one of the other accelerators discussed in Chapter 2.

3.4 Nonlinear Diffusion Acceleration

In Chapter 2, we demonstrated that nonlinear diffusion acceleration, likewise JFNK-NDA, was highly effective in accelerating the convergence of the solution of the neutron transport equation. In two spatial dimensions, we can accelerate source iteration using an identical formulation. We

begin with the multi-group formulation of the neutron transport equation

$$\hat{\Omega} \cdot \nabla \psi_g + \Sigma_{t,g} \psi_g = \frac{1}{4\pi} \left[\sum_{g'=1}^{G} \Sigma_s^{g' \to g} \phi_{g'} + Q_g \right]$$

before computing the zeroth angular moment. As in 1-D, we have

$$\begin{aligned} \phi_g &= \int_{4\pi} \psi_g \, d\hat{\Omega} \\ \vec{J}_g &= \int_{4\pi} \hat{\Omega} \, \psi_g \, d\hat{\Omega}. \end{aligned}$$

Now, unlike 1-D, the current, $\vec{J}$, is a vector. Generally, we write $\vec{J} = [J_x, J_y]^T$ in which

$$\begin{aligned} J_{x,g} &= \int_{4\pi} \mu \, \psi_g \, d\hat{\Omega} \\ J_{y,g} &= \int_{4\pi} \eta \, \psi_g \, d\hat{\Omega}. \end{aligned}$$

Given our description of the current, we can now write the zeroth angular moment of the transport equation for group g,

$$\nabla \cdot \vec{J}_g + \left(\Sigma_{t,g} - \Sigma_s^{g \to g} \right) \phi_g = \sum_{g' \neq g} \Sigma_s^{g' \to g} \phi_{g'} + Q_g. \tag{3.30}$$

Again, the neutron diffusion equation can be derived by using Fick's law for the current,

$$\vec{J}_g = -\frac{1}{3\Sigma_{t,g}} \nabla \phi_g,$$

which yields

$$\nabla \cdot \left(-\frac{1}{3\Sigma_{t,g}} \nabla \phi_g \right) + \left(\Sigma_{t,g} - \Sigma_s^{g \to g} \right) \phi_g = \sum_{g' \neq g} \Sigma_s^{g' \to g} \phi_{g'} + Q_g. \tag{3.31}$$

Just as in one spatial dimension, solving the neutron diffusion equation would be desirable over solving the neutron transport equation as all that is required is a single sparse, banded matrix inversion. However, we must recall that the neutron diffusion equation and the neutron transport equation are not consistent [18, 2]. That is, if ϕ^* is a solution to the neutron transport equation, it is not generally the solution to the neutron diffusion equation.

As in Chapter 2 and [18, 23], we supplement Fick's law with a drift term, $\hat{D}$, so that upon convergence, the augmented diffusion equation is consistent with the neutron transport

equation. We write

$$\vec{J}_g = -\frac{1}{3\Sigma_{t,g}}\nabla\phi_g + \hat{D}_g\phi_g. \tag{3.32}$$

This vector equation can be broken down into its two components,

$$J_{x,g} = -\frac{1}{3\Sigma_{t,g}}\frac{d\phi_g}{dx} + \hat{D}_{x,g}\phi_g \tag{3.33}$$

$$J_{y,g} = -\frac{1}{3\Sigma_{t,g}}\frac{d\phi_g}{dy} + \hat{D}_{y,g}\phi_g. \tag{3.34}$$

Substituting this definition of the group current (3.32) into Equation 3.30 yields a consistent diffusion equation,

$$\nabla\cdot\left(-\frac{1}{3\Sigma_{t,g}}\nabla\phi_g + \hat{D}_g\phi_g\right) + \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g = \sum_{g'\neq g}\Sigma_s^{g'\to g}\phi_{g'} + Q_g \tag{3.35}$$

or

$$\begin{aligned}\frac{d}{dx}\left[-\frac{1}{3\Sigma_{t,g}}\frac{d\phi_g}{dx} + \hat{D}_{x,g}\phi_g\right] &+ \frac{d}{dy}\left[-\frac{1}{3\Sigma_{t,g}}\frac{d\phi_g}{dy} + \hat{D}_{y,g}\phi_g\right] \\ &+ \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g = \sum_{g'\neq g}\Sigma_s^{g'\to g}\phi_{g'} + Q_g.\end{aligned} \tag{3.36}$$

In general, writing this equation in detail is too cumbersome, so we will write the system of G equations in operator notation,

$$\mathcal{D}\Phi = \left(\mathcal{S}_L + \mathcal{S}_U\right)\Phi + \vec{Q}, \tag{3.37}$$

in which

$$\begin{aligned}\mathcal{D}_g\Phi &= \nabla\cdot\left(-\frac{1}{3\Sigma_{t,g}}\nabla\phi_g + \hat{D}_g\phi_g\right) + \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g, \\ \mathcal{S}_{L,g}\Phi &= \sum_{g'<g}\Sigma_s^{g'\to g}\phi'_g, \\ \mathcal{S}_{U,g}\Phi &= \sum_{g'>g}\Sigma_s^{g'\to g}\phi'_g.\end{aligned}$$

3.4.1 Picard NDA

As we've done previously, we'll represent that transport equation using operator notation

$$\mathcal{L}\Psi = \frac{1}{4\pi}\left[(\mathcal{S}_U + \mathcal{S} + \mathcal{S}_L)\Phi + \vec{Q}\right]$$

so that we can represent the action of a transport sweep as an inversion of $\mathcal{L}$. That is, we'll write

$$\Psi = \frac{1}{4\pi}\mathcal{L}^{-1}\left[(\mathcal{S}_U + \mathcal{S} + \mathcal{S}_L)\Phi + \vec{Q}\right]$$

Now, we describe Picard NDA in Algorithm 3.4.1

Algorithm 3.4.1: Picard NDA

Input $\Phi^{(0)}$ and loop counter $n = 0$.

while Scattering source not converged **do**

Update counter, $n = n + 1$.

Execute transport sweep:

$$\Psi^{HO} = \frac{1}{4\pi}\mathcal{L}^{-1}\left[(\mathcal{S}_U + \mathcal{S} + \mathcal{S}_L)\Phi^{(n)} + \vec{Q}\right]$$

Update high-order moments:

$$\begin{aligned}\Phi^{HO} &= \int_{4\pi} \Psi^{HO}\, d\hat{\Omega} \\ \vec{J}^{HO} &= \int_{4\pi} \hat{\Omega}\, \Psi^{HO}\, d\hat{\Omega}\end{aligned}$$

Compute $\hat{D}^{(n+1)} = \hat{D}(\Phi^{HO}, \vec{J}^{HO})$.

Solve the NDA Low-Order problem, Equation 3.37 for $\Phi^{(n+1)}$:

$$\mathcal{D}^{(n+1)}\Phi^{(n+1)} = (\mathcal{S}_L + \mathcal{S}_U)\,\Phi^{(n+1)} + \vec{Q}$$

end while

As in 1-D, we see that Picard NDA is very effective compared to source iteration in solving problems with high scattering ratios. Let us consider a simple test problem given by Table 3.1.

Table 3.1: Test Problem for Convergence Comparisons

Groups	Σ_s	Σ_t	τ_x	τ_y	N_x	N_y
1	9.95	10	1	1	50	50

We'll analyze the convergence of Picard NDA and Source Iteration by perturbing the solution

to this problem and using it as the initial iterate for each algorithm. This initial iterate will be given by

$$\Phi^{(0)}(x,y) = \Phi^*(x,y) + .1\left[\sin(20\pi x/\tau_x) + \sin(20\pi y/\tau_y) + \sin(\pi x/\tau_x) + \sin(\pi y/\tau_y)\right].$$

This represents a high frequency error mode along with a low frequency error mode. We'll compare the convergence of Source Iteration and Picard NDA using $\Phi^{(0)}$ as an initial iterate. Figures 3.13 and 3.14 plot errors over the course of several iterations.

These figures clearly demonstrate how effective Picard NDA can be over Source Iteration. In Figure 3.13, we can see that after 50 source iterations, the infinity norm of the error has only been decreased by a single order of magnitude. Source iteration has effectively damped the high frequency error mode, however the low frequency mode still persists. After another 50 iterations, the low frequency error mode has been damped further, but still exists.

In Figure 3.14 we see that Picard NDA handles both the high and low frequency error modes effectively in relatively few iterations. After 1 iteration, the error has been lowered by an entire order of magnitude. Each additional iteration reduces the error by roughly a single order of magnitude. After 5 iterations the maximum error is less than 10^{-5}. After 10 iterations the maximum error is roughly 1.3×10^{-9}. Compare this to source iteration, in which the maximum error after 10 iterations is roughly 1.3×10^{-1}.

These results are nearly identical to the results produced in Chapter 2.

3.4.2 JFNK-NDA

As in Chapter 2, we can cast the NDA algorithm as a system of nonlinear equations and use a Jacobian-Free Newton-Krylov method to solve the system. In this case, we write the system as

$$F(\Phi) \equiv [F_1(\Phi)\ F_2(\Phi)\ \cdots F_G(\Phi)]^T = 0, \tag{3.38}$$

in which

$$F_g(\Phi) = \left[\mathcal{D}_g\Phi^{(n+1)} - (\mathcal{S}_{L,g} + \mathcal{S}_{U,g})\right]\Phi^{(n+1)} + Q_g. \tag{3.39}$$

We remind the reader that

$$\begin{aligned}
\mathcal{D}_g\Phi &= \nabla\cdot\left(-\frac{1}{3\Sigma_{t,g}}\nabla\phi_g + \hat{D}_g(\Phi)\phi_g\right) + \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g, \\
\mathcal{S}_{L,g}\Phi &= \sum_{g'<g}\Sigma_s^{g'\to g}\phi_{g'}, \\
\mathcal{S}_{U,g}\Phi &= \sum_{g'>g}\Sigma_s^{g'\to g}\phi_{g'},
\end{aligned}$$

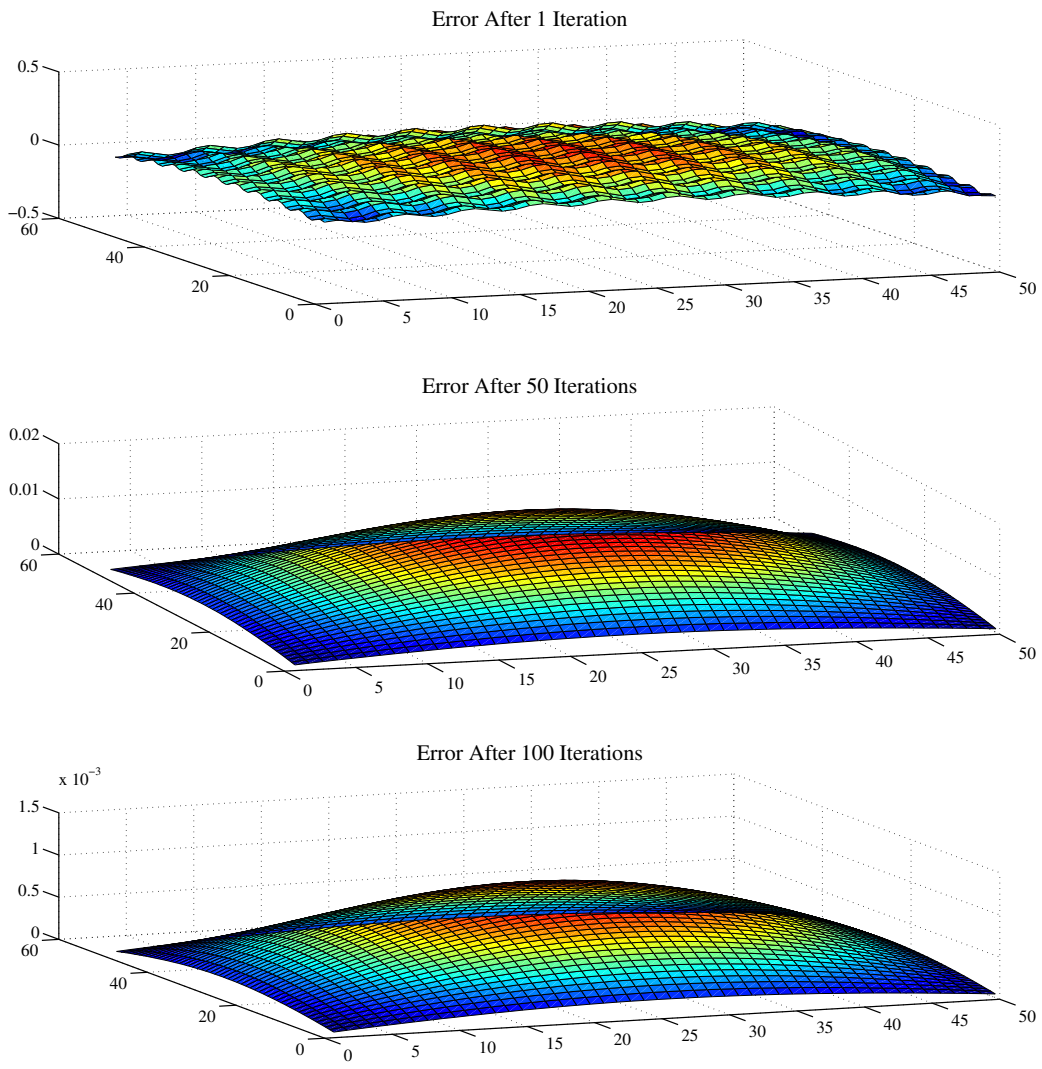

Figure 3.13: Source iteration damps high frequency error modes quickly, however low frequency errors persist.

in which the nonlinearity in F is due to the dependence of $\hat{D}$ on Φ.

We evaluate F in the following order:

1. Input Φ.

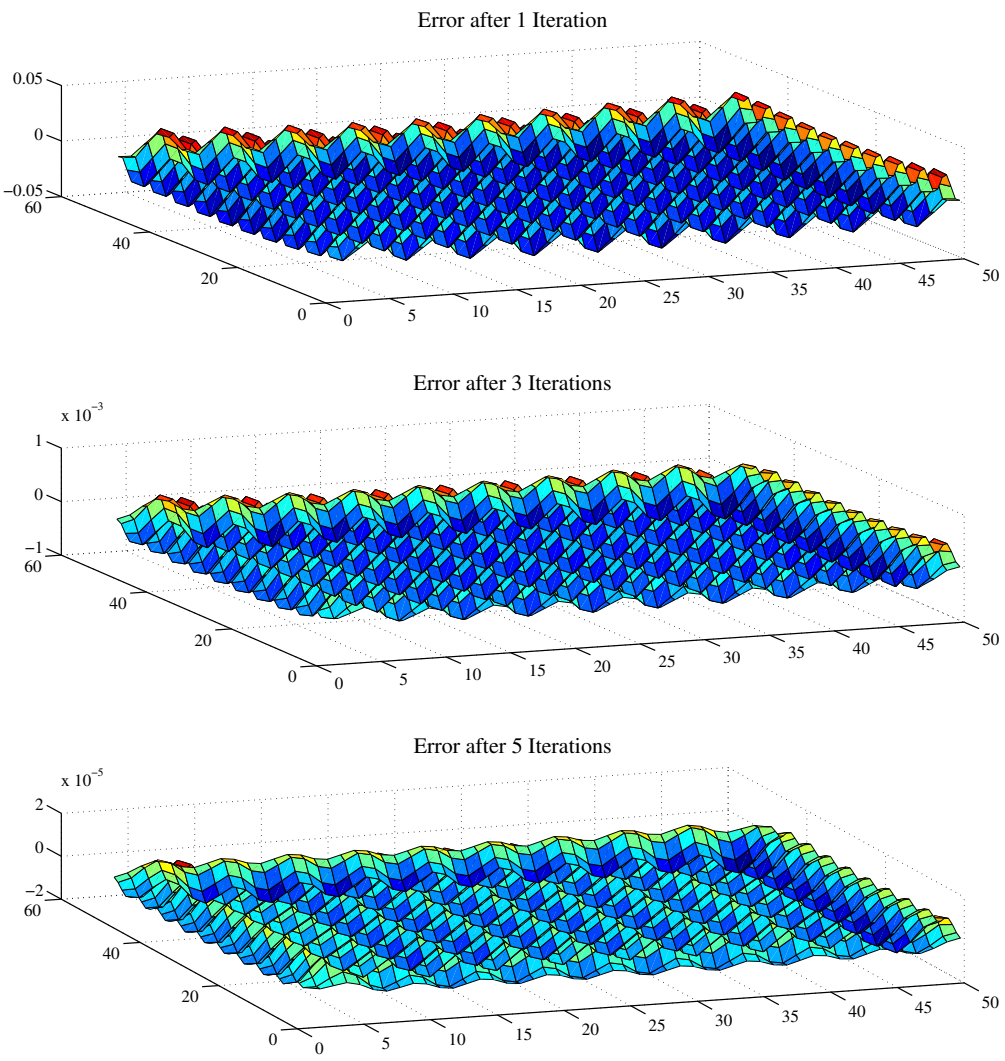

Figure 3.14: NDA damps both high and low frequency error modes quickly.

2. Execute a single transport sweep, recover Ψ^{HO}.

3. Compute high-order moments Φ^{HO} and $\vec{J}^{HO}$.

4. Compute $\hat{D} = \hat{D}(\Phi^{HO}, \vec{J}^{HO})$.

5. Compute $F(\Phi)$.

We solve $F(\Phi) = 0$ using Newton-GMRES. That is, given an initial iterate, $\Phi^{(0)}$, we iterate

$$\Phi^{(n+1)} = \Phi^{(n)} - F'(\Phi^{(n)})^{-1} F(\Phi^{(n)})$$

where the Newton step is computed by applying GMRES to the linear problem

$$F'(\Phi^{(n)})^{-1} s = -F(\Phi^{(n)}).$$

We precondition the linear solve with the matrix M^{-1}, where M is given by

$$M = \mathcal{D} - (\mathcal{S}_L + \mathcal{S}_U)$$

where $\mathcal{D}$ is evaluated using the $\hat{D}$ from the most recent evaluation F.

3.5 Test Problems

As in Chapter 2, we'll compare the methods we've discussed for solving the 2-D transport equation on a set of test problems. These three tests are summarized in Table 3.2. Each of these test problems is set in two groups and the domain is comprised of a single material. For tests 1 and 2, we'll use vacuum boundary conditions on each boundary. For test 3, we'll consider a reflective boundary along the x- and y-axes.

We'll apply source iteration, GMRES, Picard NDA and JFNK-NDA to each of these problems and compare the iteration statistics. For each test, we'll use an S_8 angular discretization. Furthermore, we will abandon the diamond difference discretization for the step characteristics discretization in space.

Table 3.3 demonstrates the convergence behavior for the test problems described in Table 3.2 for four methods; Source Iteration, GMRES, JFNK-NDA, and Picard NDA. As we can see, as the scattering ratio increases, source iteration becomes more and more inefficient. For Test 3, it requires over 2500 transport sweeps to reduce the residual to a tolerance of 10^{-8}. GMRES reaches the same convergence criterion in only 134 iterations. For all 3 test problems, both Picard NDA and JFNK-NDA perform very well, converging in under 20 iterations for each test problem.

3.6 Conclusions

Moving forward, our goal will be to compute the dominant eigenvalue of the transport equation, known as k_{eff}, or k-effective. The results from Chapters 2 and 3 will influence our choice in

Table 3.2: Test Problem for Convergence Comparisons

	Tests					
Parameter	1		2		3	
Group	1	2	1	2	1	2
Σ_t	.5	2	1	3	5	10
$\Sigma_s^{1\to g}$	0.40	0.05	0.90	0.10	4.00	1.00
$\Sigma_s^{2\to g}$	0.00	1.95	0.50	2.40	0.00	9.95
τ_x	5		1		10	
τ_y	5		5		10	
N_x	50		50		100	
N_y	50		50		100	

Table 3.3: Transport Sweeps to Convergence

	Tests		
Method	1	2	3
Source Iteration	200	55	2506
GMRES	19	15	134
Picard NDA	11	11	11
JFNK-NDA	16	14	14

algorithms for solving the k-eignevalue problem. We have seen how effective NDA-based algorithms can be for accelerating the fixed-source problem. We will demonstrate in the following chapter how NDA can be used in a very similar manner to accelerate the convergence of the eigenvalue calculation.

Much like we did with the fixed-source problem, we can use NDA for the eigenvalue problem to shift a large portion of the work from high-order transport sweeps to low-order diffusion equation solves. This leads to two eigenvalue algorithms, NDA-PI and NDA-NCA, both of which will be discussed at length in the following chapter.

Chapter 4

Criticality Calculations

All of the work performed in Chapters 2 and 3 has been in preparation for this chapter, in which we explore numerical methods for computing k_{eff}, or the dominant eigenvalue of the neutron transport equation. Calculation of k_{eff} can be difficult and computationally expensive, especially for practical reactor configurations, in which the dominance ratio is very near to 1. If we order the eigenvalues so that $|k_i| > |k_j|$ for all $i < j$, the *dominance ratio*, ρ, is defined to be the ratio of the magnitude largest eigenvalue ($k_{eff} = k_1$) to the magnitude of the second largest eigenvalue [17, 23]. That is,

$$\rho = \frac{|k_2|}{|k_{eff}|}.$$

In the previous chapters, we've developed methods to solve

$$(\mathcal{I} - \mathcal{K})\,\Phi = g,$$

where the right hand side, g, represents some fixed quantity which is a function of the material cross-sections, boundary conditions and fixed source. In both Chapters 2 and 3, we found that NDA-based methods performed very well for a wide range of problems and was able to speed up the convergence by several orders of magnitude over source iteration.

In this chapter, we will consider methods for solving the k-eigenvalue problem, which consists of computing the dominant eigenvalue for

$$(\mathcal{I} - \mathcal{K})\,\Phi = \frac{1}{k}\mathcal{F}\Phi. \tag{4.1}$$

In Equation 4.1, $\mathcal{F}\Phi$ represents the fission source term, which is scaled by $\frac{1}{k_{eff}}$ to enforce that the number of fission neutrons in each generation remains constant. For the k-eigenvalue problem, the analog of source iteration is called *power iteration*. Just like source iteration for

the fixed-source problem, power iteration is the most basic method for computing the dominant eigenvalue. We'll discuss power iteration in the following section, however before we move on, it is important to note that power iteration suffers from the same deficiencies that source iteration does for the fixed source problem, along with additional difficulties that arise in the eigenvalue problem.

In Chapter 2, we showed that for highly scattering problem ($\Sigma_s/\Sigma_t \approx 1$) source iteration may take thousands or tens of thousands of iterations to converge. This is due to source iteration's inability to quickly damp long-wave length (diffusion) error modes. Essentially, for highly scattering problems, source iteration cannot quickly compute the scattering source.

In this chapter, we find that power iteration struggles in the same regime. For high scattering ratios, power iteration is slow to converge the scattering source. In addition to the scattering source, power iteration must converge the fission production source. For problems in which the dominance ratio, ρ, is nearly 1, power iteration will be slow to converge the eigenvalue.

For this reason, it is important that we develop methods that allow us to accelerate the convergence of the dominant eigenvalue and its associated eigenvector. Keeping in mind the noted deficiencies of power iteration, these methods must accelerate the computations of both the scattering and fission source terms. We've shown how NDA-based methods can accelerate the scattering source term and in this chapter, we will demonstrate how we can use similar methods to accelerate the eigenvalue computation. We'll derive a sequence of consistent low-order eigenvalue problems which, upon convergence, will yield the dominant eigenvalue and can be iterated upon at a fraction of the cost of a transport sweep.

Recent works have investigated using Newton's method for solving the eigenvalue problem for both neutron transport and diffusion theory [11, 12, 17, 23, 33, 31]. In [12] Newton's method is applied directly transport equation, whereas in [23, 33, 31] Newton's method is applied to an angular moment-based system. In [4], the authors use Nonlinear Krylov Acceleration in an attempt to accelerate the basic power iteration algorithm.

4.1 Power Iteration

In the well-known and understood power method from mathematics, one considers successive applications of a matrix, A, to an approximation to the eigenvector associated with the dominant eigenvalue,

$$\begin{aligned} \vec{y}_{n+1} &= A\vec{x}_n \\ \vec{x}_{n+1} &= \frac{\vec{y}_{n+1}}{\|\vec{y}_{n+1}\|}. \end{aligned}$$

Under the appropriate conditions (see Appendix B), $\|\vec{y}_{n+1}\|$ converges to the dominant eigenvalue and $\vec{x}_{n+1}$ converges to its associated eigenvector. The power iteration algorithm is very similar in the way in which we compute k_{eff}.

As a reminder, we're trying to compute the maximum value of k for which

$$(\mathcal{I} - \mathcal{K})\,\Phi = \frac{1}{k}\mathcal{F}\Phi \tag{4.2}$$

is satisfied for some nonzero scalar flux Φ. We can rewrite this as

$$(\mathcal{I} - \mathcal{K})^{-1}\,\mathcal{F}\Phi = k\Phi,$$

which suggests an iterative procedure similar to the one defined above

$$\begin{aligned} \Xi_{n+1} &= (\mathcal{I} - \mathcal{K})^{-1}\,\mathcal{F}\Phi_n \\ \Phi_{n+1} &= \frac{\Xi_{n+1}}{\|\Xi_n\|}. \end{aligned} \tag{4.3}$$

However, upon further investigation, we realize that this iterative process would involve the inversion of $(\mathcal{I} - \mathcal{K})$ at each step (Equation 4.3). The inversion of this operator is the same as solving a fixed source problem in which the right hand side is given by the fission source. In general, this is an inefficient process and more expensive than necessary.

Instead, we avoid completely inverting $(\mathcal{I} - \mathcal{K})$. Instead, we execute some fixed number of transport sweeps for a fixed right-hand side, approximating (to some degree) the inversion of $(\mathcal{I} - \mathcal{K})$. This is described mathematically in Algorithm 4.1, referred to as Power Iteration.

The condition for exiting the inner while loop in Algorithm 4.1 can be replaced by some condition which measures the relative difference between successive iterates. In practice, either method will yield the appropriate eigenvalue, though depending on the choice of N or the relative tolerance, one instance may outperform the other. However, this fact is somewhat uninteresting as we have already noted that the convergence of power iteration is unacceptably slow and demands acceleration.

In Section 4.3, we will demonstrate that power iteration may take thousands of transport sweeps to compute the eigenvalue correct to six decimal places for a realistic reactor-like configuration. We'll also demonstrate that the same accuracy solution can be obtained using an NDA-based method using on order 10 transport sweeps and a series of less expensive diffusion solves.

Algorithm 4.1: Power Iteration

Choose initial iterate ϕ^0, approximate eigenvalue k^0, outer iteration counter $m = 0$
while eigenvalue not converged **do**
$m = m + 1$
Set $\Phi^0 = \phi^{m-1}$, inner iteration counter $n = 0$
while $n < N$ **do**
$n = n + 1$
Execute transport sweep:

$$\begin{aligned} \Psi^{(n)} &= \mathcal{L}^{-1}\left[(\mathcal{S}_U + \mathcal{S}_D)\Phi^{(n-1)} + \mathcal{S}_L\Phi^{(n)} + \frac{1}{k^{m-1}}\mathcal{F}\phi^{(m-1)}\right] \\ \Phi^{(n)} &= \int \Psi^{(n)} d\hat{\Omega} \end{aligned}$$

end while
Update approximate eigenvalue

$$k^m = k^{m-1}\frac{\int \nu\Sigma_f\Phi^{(n)}dV}{\int \nu\Sigma_f\phi^{(m-1)}dV}$$

and the flux for the fission term

$$\phi^{(m)} = \Phi^{(n)}$$

end while

4.2 NDA-Based Eigenvalue Acceleration

Consider the multi-group k-eigenvalue problem

$$\hat{\Omega}\cdot\nabla\psi_g + \Sigma_{t,g}\psi_g = \frac{1}{4\pi}\left[\sum_{g'=1}^{G}\Sigma_s^{g'\to g}\phi_{g'} + \frac{\chi_g}{k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\phi_{g'}\right] \tag{4.4}$$

where $g = 1, 2, \ldots, G$. Just as we did in Chapters 2 and 3 when deriving the NDA low-order equation, let us compute the 0^{th} angular moment of Equation 4.4,

$$\nabla\cdot\vec{J}_g + \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g = \sum_{g'\neq g}\Sigma_s^{g'\to g}\phi_{g'} + \frac{\chi_g}{k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\phi_{g'}. \tag{4.5}$$

Again, we must close this equation by relating the current to the scalar flux, and we do so using the same Fick's law plus correction term description we've used throughout this thesis

[23, 18, 32],

$$\vec{J}_g = -\frac{1}{3\Sigma_{t,g}}\nabla\phi_g + \hat{D}_g\phi_g. \tag{4.6}$$

Combining Equations 4.5 and 4.6 yields the NDA low-order eigenvalue problem,

$$\nabla\cdot\left[-\frac{1}{3\Sigma_{t,g}}\nabla\phi_g + \hat{D}_g\phi_g\right] + \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g = \sum_{g'\neq g}\Sigma_s^{g'\to g}\phi_{g'} + \frac{\chi_g}{k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\phi_{g'}. \tag{4.7}$$

Upon convergence, the high-order eigenvalue problem (Equation 4.4) and the low-order eigenvalue problem (Equation 4.7) have the same dominant eigenvalue. That is, if $\hat{D}$ is evaluated using the dominant eigenvector and the associated high-order current, then Equation 4.7 has the same dominant eigenvalue/eigenvector pair as Equation 4.4.

In the two following subsections, we'll discuss two NDA-based eigenvalue methods, NDA-PI and NDA-NCA [23]. Both of these methods make use of the same low-order eigenvalue equation, however they differ in the way this equation in solved. The general idea is similar to that for fixed-source problems. We use the high-order equation to update the correction term, $\hat{D}$. Given this updated $\hat{D}$, we use can use the low-order equation to update the approximate eigenvector and eigenvalue.

4.2.1 NDA-PI

The first high-order/low-order eigenvalue method is referred to as NDA accelerated Power Iteration, or NDA-PI [23]. This aptly named method uses the transport eigenvalue equation (Equation 4.4) to build compute a correction term, $\hat{D}$, for each group. Using this $\hat{D}$, we build a system of low-order equations (Equation 4.7) and solve this system using the power iteration technique described before. We present this method formally in Algorithm 4.2.1.

NDA-PI improves upon power iteration in two distinct ways. First, we have accelerated the convergence of the scattering source by using NDA. Much like NDA for the fixed-source problem, NDA-PI allows us to converge the scattering source in significantly fewer iterations. Secondly, now the power iterations take place in the low-order problem instead of in high-order space. Not only have we cut down on the total number of iterations, but much of the work has been moved to the low-order eigenvalue problem. This is much less expensive to solve and saves us a considerable amount of high-order work.

In Section 4.3 we demonstrate the convergence results for NDA-PI for a set of 1-D and 2-D test problems. We'll notice that NDA-PI can be extremely effective compared to standard PI, however there is still room for improvement by cutting down the number of low-order diffusion solves. We'll do this in two ways; In Section 4.2.2 we'll consider NDA-NCA in which the low-

Algorithm 4.2.1: NDA-PI

Choose initial iterate Φ^0, approximate eigenvalue k^0, outer iteration counter $m = 0$

while transport eigenvalue not converged **do**

$m = m + 1$

Execute a single transport sweep and compute angular moments

$$\begin{aligned}\Psi^{HO} &= \mathcal{L}^{-1}\left[\mathcal{S}\Phi^{(m-1)} + \frac{1}{k^{m-1}}\mathcal{F}\Phi^{(m-1)}\right] \\ \Phi^{HO} &= \int \Psi^{HO} d\hat{\Omega} \\ \Phi^{HO} &= \int \hat{\Omega}\Psi^{HO} d\hat{\Omega}\end{aligned}$$

Compute $\hat{D}$ for each group

$$\hat{D}_g = \frac{\vec{J}_g^{HO} + \frac{1}{3\Sigma_{t,g}}\nabla\Phi_g^{HO}}{\Phi_g^{HO}}$$

Set inner iteration counter $n = 0$ and $\phi^{(0)} = \Phi^{HO}$, $K^0 = k^{m-1}$.

while diffusion eigenvalue not converged **do**

$n = n + 1$

Update diffusion eigenvalue and eigenvector

$$\begin{aligned}\phi^{(n)} &= \left[\mathcal{D} - (\mathcal{S}_L + \mathcal{S}_U)\right]^{-1}\frac{1}{K^{n-1}}\mathcal{F}\phi^{(n-1)} \\ K^n &= K^{n-1}\frac{\int \nu\Sigma_f\phi^{(n)}dV}{\int \nu\Sigma_f\phi^{(n-1)}dV}\end{aligned}$$

end while

Update transport eigenvalue

$$\begin{aligned}\Phi^{(m)} &= \phi^{(n)} \\ k^m &= K^n\end{aligned}$$

end while

order problem is solved with Newton's method. However, first we will consider a numerical technique for accelerating the convergence of the eigenvalue.

Wielandt Shift

For any of these algorithms, when the dominance ratio, ρ, is nearly 1 we will suffer from slower convergence. As mentioned earlier, the power method converges with $\mathcal{O}\left(\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^m\right)$. For reactor

criticality problems, ρ can be very close to 1 and convergence can be unacceptably slow. We can use known acceleration techniques to lower the dominance ratio and speed up convergence. We'll consider one of these methods, the Wielandt shift [17, 23, 22, 26], in detail in this section.

In Algorithms 4.2.1 we move most of the computational effort to solving the low-order problem

$$\mathcal{L}\Phi = \lambda\mathcal{F}\Phi. \tag{4.8}$$

for a new approximation to the eigenvalue and eigenvector. Note that $\mathcal{L}$ is not the same operator as was defined in the previous chapter. For now, we will assume that $\Sigma_f(x) > 0$ for all x so that $\mathcal{F}$ is nonsingular.

We may re-write Equation 4.8 as

$$\Phi = \lambda\mathcal{R}\Phi \tag{4.9}$$

where

$$\Phi = \mathcal{F}\phi \tag{4.10}$$

$$\mathcal{R} = \mathcal{F}\mathcal{L}^{-1}. \tag{4.11}$$

Since both $\mathcal{F}$ and $\mathcal{L}$ are nonsingular, we know that $\mathcal{R}$ is nonsingular and has no zero eigenvalues. We can list the eigenvalues

$$\lambda_1 < |\lambda_2| < \cdots < |\lambda_N|. \tag{4.12}$$

Now, suppose we have some approximation $\tilde{\lambda}$ to λ_1 such that

$$|\lambda_1 - \tilde{\lambda}| < |\lambda_2 - \tilde{\lambda}| < \cdots < |\lambda_N - \tilde{\lambda}|. \tag{4.13}$$

We can modify the original eigenvalue problem and solve

$$(\mathcal{L} - \tilde{\lambda}\mathcal{F})\phi = \mu\mathcal{F}\phi \tag{4.14}$$

for the eigenvalue μ. Again, we re-write Equation 4.14 as

$$\Phi = \mu\tilde{\mathcal{R}}\Phi \tag{4.15}$$

where

$$\Phi = \mathcal{F}\phi \tag{4.16}$$

$$\tilde{\mathcal{R}} = \mathcal{F}(\mathcal{L} - \tilde{\lambda}\mathcal{F})^{-1}. \tag{4.17}$$

Since $\tilde{\lambda}$ is not in the spectrum of $\mathcal{R}$, $\tilde{\mathcal{R}}$ is still nonsingular.

The power method now converges to $\mu = \lambda_1 - \tilde{\lambda}$ at a rate of

$$\tilde{\sigma} = \frac{\lambda_1 - \tilde{\lambda}}{\lambda_2 - \tilde{\lambda}}. \tag{4.18}$$

Clearly, this method can converge at a much faster rate than the standard power method. This method for "shifting" eigenvalues is known as the Wielandt shift [17, 23].

In practice, we want to choose $\tilde{\lambda}$ close to λ_1, but not so close that $\mathcal{R}$ nears singularity. In [22], the value of $\tilde{\lambda}$ is chosen so that

$$\tilde{\lambda} = \lambda_1^{(k)} - (\text{small positive constant})$$

where $\lambda_1^{(k)}$ is the current best approximation to the dominant eigenvalue. In [22], the small positive constant is chosen to be .01, and, in practice, we find this value to be quite effective.

For 1-D problems, the Wielandt shift can be highly effective in reducing the number of low-order diffusion solves as we will see in the 1-D numerical results. However, by shifting the spectrum, the Wielandt shift makes the problem less diagonally dominant and more ill-conditioned [17]. This can easily be seen by considering the above analysis and Equation 4.17 and Inequality 4.13. This ill-conditioning is not a major concern in a 1-D, 1-group problem, as the inversion of $\bar{\mathcal{R}}$ is done with direct methods. However, when the problem grows larger and we are forced to move from direct methods to iterative methods, this ill-conditioning can prove to increase the amount of iterative work for each matrix inversion. For this reason, we'll present results in 1-D, however we will not consider the Wielandt shift in higher dimensions.

4.2.2 NDA-NCA

The second eigenvalue acceleration technique we'll discuss is NDA accelerated Nonlinear Criticality Acceleration [23], or NDA-NCA. We begin by considering the same low-order eigenvalue equation that was derived for NDA-PI,

$$\nabla \cdot \left[-\frac{1}{3\Sigma_{t,g}} \nabla \phi_g + \hat{D}_g \phi_g \right] + \left(\Sigma_{t,g} - \Sigma_s^{g \to g} \right) \phi_g = \sum_{g' \neq g} \Sigma_s^{g' \to g} \phi_{g'} + \frac{\chi_g}{k_{eff}} \sum_{g'=1}^{G} \nu \Sigma_{f,g'} \phi_{g'}. \tag{4.19}$$

Now, for each group, we can write a nonlinear system of equations which is identically zero when Equation 4.19 has been solved. This nonlinear equation is given by

$$\begin{aligned} F_g(\Phi, k) &= \nabla \cdot \left[-\frac{1}{3\Sigma_{t,g}} \nabla \phi_g + \hat{D}_g \phi_g \right] + \left(\Sigma_{t,g} - \Sigma_s^{g \to g} \right) \phi_g - \sum_{g' \neq g} \Sigma_s^{g' \to g} \phi_{g'} - \frac{\chi_g}{k} \sum_{g'=1}^{G} \nu \Sigma_{f,g'} \phi_{g'}. \\ &= \mathcal{D}_g \phi_g - \left(\mathcal{S}_U + \mathcal{S}_L \right) \Phi - \frac{1}{k} \chi_g \mathcal{F} \Phi \end{aligned} \tag{4.20}$$

Combining these G nonlinear systems and a constraint equation for k, we build a square system of nonlinear equations, $F(\Phi, k)$, given by

$$F(\Phi, k) = \left[F_1(\Phi, k) \; F_2(\Phi, k) \; \ldots \; F_G(\Phi, k) \; F_k(\Phi, k) \right]^T$$

where

$$F_k(\Phi, k) = k - \int \mathcal{F} \Phi dV \tag{4.21}$$

as done in [23, 17].

However, instead of solving $F(\Phi, k) = 0$, we can use nonlinear elimination to remove the dependence of F on k. We notice Equation 4.21 is satisfied only when

$$k = \int \mathcal{F} \Phi dV. \tag{4.22}$$

Therefore, we write a new system of equations $F(\Phi) = 0$, given by

$$F(\Phi) = \left[F_1(\Phi) \; F_2(\Phi) \; \ldots \; F_G(\Phi) \right]^T$$

where the eigenvalue, k, in each F_g is a function of Φ as described in Equation 4.22 [17, 23]. We present this method as Algorithm 4.2.2.

The goal with NDA-NCA is that by wrapping the low-order eigenvalue calculation in Newton's method we can reduce the amount of low-order work required in NDA-PI. In Section 4.3 we will see that the savings can be vast.

Using Newton's Method to Solve the Low-Order Eigenvalue Problem

In [17, 23], the authors elect to use a Jacobian-Free Newton-Krylov method to solve $F(\Phi) = 0$. With this method, the Newton step, w, is computed by approximately solving

$$F'(\Phi) w = -F(\Phi)$$

Algorithm 4.2.2: NDA-NCA

Choose initial iterate $\Phi^{(0)}$, approximate eigenvalue k^0, outer iteration counter $m = 0$

while eigenvalue not converged **do**

Update counter, $m = m + 1$.

Execute transport sweep and compute consistency term:

$$\begin{aligned}
\Psi^{(m)} &= \mathcal{L}^{-1}\left[\mathcal{S}\Phi^{(m-1)} + \frac{1}{k^{m-1}}\chi\mathcal{F}\Phi^{(m-1)}\right] \\
\Phi^{HO} &= \int \Psi^{(m)} d\hat{\Omega}, \quad \vec{J}^{HO} = \int \hat{\Omega}\Psi^{(m)} d\hat{\Omega} \\
\hat{D}^{(m)} &= \frac{\vec{J}^{HO} + \frac{1}{3\Sigma_t}\nabla\Phi^{HO}}{\Phi^{HO}}
\end{aligned}$$

Solve $F(\phi) = 0$ using a Jacobian-Free Newton-Krylov method, where

$$\begin{aligned}
F(\phi) &= \mathcal{D}\phi - (\mathcal{S}_U + \mathcal{S}_L)\phi - \frac{1}{k_{eff}}\chi\mathcal{F}\phi \text{ in which} \\
k_{eff} &= \int \mathcal{F}\phi \, dV
\end{aligned} \tag{4.23}$$

Update flux and eigenvalue: $\Phi^{(m)} = \phi$, $k^{(m)} = \int \nu\Sigma_f\phi \, dV$.

end while

using GMRES. Here, the Jacobian $F'(\Phi)$ is not formed explicitly. Instead, we write a Jacobian-vector product routine which approximately computes the action of the Jacobian on a vector, given by

$$F'(\Phi)w \approx \frac{F(\Phi + \epsilon w) - F(\Phi)}{\epsilon}$$

for some small ϵ. This finite-difference Jacobian-vector is very commonly used in Jacobian-Free Newton-Krylov implementations, however may be suboptimal.

In addition, since GMRES is being used to solve for the Newton step, preconditioning becomes very important. The authors of [17, 23] use a single diffusion power iteration as a preconditioner and find that this is highly effective. We will demonstrate this for a variety of test problems in Section 4.3.

However, instead of using this finite difference Jacobian-vector product, we can demonstrate that the Jacobian is a rank-one update to a sparse, banded matrix. This allows us to employ the Sherman-Morrison formula [15] for a less expensive, direct inverse. This fact is demonstrated in the following theorem [31].

Theorem 3. *The Jacobian, $F'(\phi)$, for the function, F, described by Equation 4.23, can be*

decomposed into the sum of a sparse matrix and a rank-one update.

Proof. We can compute the action of the Jacobian on a vector using the formula

$$F'(\phi)w = \lim_{\epsilon \to 0} \frac{F(\phi + \epsilon w) - F(\phi)}{\epsilon}. \tag{4.24}$$

Before applying this formula, we should note that we can break down the function, F, into its linear and nonlinear components

$$\begin{aligned} F(\phi) &= \mathcal{A}\phi - \mathcal{N}(\phi) \text{ where} & (4.25)\\ \mathcal{A}\phi &= \mathcal{D}\phi - (\mathcal{S}_U + \mathcal{S}_L)\phi & (4.26)\\ \mathcal{N}(\phi) &= \frac{1}{k_{eff}(\phi)}\chi\mathcal{F}\phi & (4.27) \end{aligned}$$

In Equation 4.27 we write $k_{eff}(\phi)$ to note the explicit dependence of k_{eff} on ϕ. It is important to note that k_{eff} is a linear function of ϕ.

Now, we begin by noticing that we can write

$$F'(\phi)w = \mathcal{A}w + \mathcal{N}'(\phi)w. \tag{4.28}$$

At this point, the only thing left to do is compute $\mathcal{N}'(\phi)w$. For this, we turn back to Equation 4.24 and write

$$\begin{aligned} \mathcal{N}'(\phi)w &= \lim_{\epsilon \to 0} \frac{\mathcal{N}(\phi + \epsilon w) - \mathcal{N}(\phi)}{\epsilon}\\ &= \lim_{\epsilon \to 0} \frac{\frac{1}{k_{eff}(\phi+\epsilon w)}\chi\mathcal{F}(\phi + \epsilon w) - \frac{1}{k_{eff}(\phi)}\chi\mathcal{F}\phi}{\epsilon}\\ &= \lim_{\epsilon \to 0} \frac{\frac{\chi\mathcal{F}\phi + \epsilon\chi\mathcal{F}w}{k_{eff}(\phi)+\epsilon k_{eff}(w)} - \frac{\chi\mathcal{F}\phi}{k_{eff}(\phi)}}{\epsilon}\\ &= \lim_{\epsilon \to 0} \frac{\frac{\epsilon\chi\mathcal{F}w}{k_{eff}(\phi)+\epsilon k_{eff}(w)}}{\epsilon} + \lim_{\epsilon \to 0} \frac{\frac{\chi\mathcal{F}\phi}{k_{eff}(\phi)+\epsilon k_{eff}(w)} - \frac{\chi\mathcal{F}\phi}{k_{eff}(\phi)}}{\epsilon}\\ &= \frac{1}{k_{eff}(\phi)}\chi\mathcal{F}w + \lim_{\epsilon \to 0} \frac{\frac{\chi\mathcal{F}\phi \cdot k_{eff}(\phi) - \chi\mathcal{F}\phi \cdot \left[k_{eff}(\phi)+\epsilon k_{eff}(w)\right]}{\left[k_{eff}(\phi)+\epsilon k_{eff}(w)\right] \cdot k_{eff}(\phi)}}{\epsilon}\\ &= \frac{1}{k_{eff}(\phi)}\chi\mathcal{F}w + \lim_{\epsilon \to 0} \frac{-\chi\mathcal{F}\phi \cdot k_{eff}(w)}{[k_{eff}(\phi) + \epsilon k_{eff}(w)] \cdot k_{eff}(\phi)}\\ &= \frac{1}{k_{eff}(\phi)}\chi\mathcal{F}w - \frac{k_{eff}(w)}{k_{eff}(\phi)^2}\chi\mathcal{F}\phi \end{aligned} \tag{4.29}$$

Lastly, let's show that we can write the discrete form of the final term of Equation 4.29 as a

rank-one matrix times the vector w. In order to see this, we should first note that we can write

$$k_{eff}(\vec{\phi}) = \vec{W}^T F \vec{\phi} \tag{4.30}$$

in which $\vec{W}$ is a vector of weights used to compute the numerical integral over the spatial domain.

Now, we have

$$\begin{aligned} -\frac{k_{eff}(\vec{w})}{k_{eff}(\vec{\phi})^2}\chi F\vec{\phi} &= -\frac{\vec{W}^T F\vec{w}}{(\vec{W}^T F\vec{\phi})^2}\chi F\vec{\phi} \\ &= -\frac{\chi F\vec{\phi}(\vec{W}^T F\vec{w})}{c^2} \\ &= -\frac{(\chi F\vec{\phi}\vec{W}^T F)\vec{w}}{c^2} \\ &= (\vec{y}\vec{z}^T)\vec{w} \end{aligned}$$

where

$$\begin{aligned} \vec{y} &= -\frac{\chi F\vec{\phi}}{c} \\ \vec{z} &= \frac{\vec{W}^T F}{c} \\ c &= \vec{W}^T F\vec{\phi}. \end{aligned}$$

Combining these results, we have

$$\begin{aligned} J\vec{w} &= L\vec{w} + \frac{1}{c}F\vec{w} + \vec{y}\vec{z}^T\vec{w} \\ &= M\vec{w} + \vec{y}\vec{z}^T\vec{w} \\ &= (M + \vec{y}\vec{z}^T)\vec{w} \end{aligned} \tag{4.31}$$

That is, the Jacobian, J, can be decomposed into a sparse, banded matrix, M, and a rank-one update. □

This matrix, M, is the same matrix that that we use as the preconditioning operator when solving for the Newton step using GMRES. Clearly, using the Sherman-Morrison formula to invert the Jacobian costs the same as two preconditioner applications and thus must be less expensive that two GMRES iterations. This implies a nearly guaranteed savings over Newton-GMRES for solving the low-order eigenvalue problem.

Furthermore, when using GMRES to solve

$$F'(\Phi)w = -F(\Phi),$$

we do not solve this equation exactly. Instead, we find a w such that

$$\|F'(\Phi)w + F(\Phi)\| < \eta\|F(\Phi)\|,$$

in which η is some small positive constant. In contrast, using the Sherman-Morrison formula with direct inversion of the matrix, M, computes the Newton step to machine tolerance.

To summarize, by exploiting the structure of the Jacobian of F, we can compute a higher accuracy Newton step at a lower cost than solving this linear system using GMRES. In addition, it is important to note that, in practice, we find that it is most efficient to take a single Newton step instead of solving $F(\Phi) = 0$ to a tight tolerance. The low-order problem is used to update the eigenvalue/eigenvector pair, but at early stages in the iteration, the low-order problem is only a weak approximation to the high-order problem. It is counter productive to put too much effort into solving what is essentially the wrong problem. Instead, at each iteration, we take a single Newton step and use the high-order equation to build a more accurate low-order problem.

Applying a Line Search

Most implementations of Newton's method include a line-search (see Appendix A) to ensure that the new Newton iterate yields a smaller residual. With NDA-NCA, we forgo the traditional line-search which evaluates F at the new iterate. Instead, we use our knowledge that the fundamental mode is strictly positive [21, 23, 2, 7] to demand that the new scalar flux is positive everywhere. If the new flux is negative at any point in the domain, we cut back on the length of the step until the new flux is everywhere positive. In practice, we see that this "physics-based line-search" is only activated when the mesh is too coarse.

4.3 Results

In this section we will present convergence results for each of these methods in both 1 and 2 spatial dimensions. These results demonstrate that NDA-based eigenvalue methods are capable of providing a 2-3 order of magnitude speedup over power iteration. We'll first present results for three 1-D, 1 group, multimaterial eigenvalue problems as simple proof of concept. After this, we'll look at three 2-D, 2 group, multi-material test problems that demonstrate the effectiveness of these algorithms on more realistic reactor problems.

4.3.1 1D Test Problems and Results

In order to draw a comparison between the effectiveness of these algorithms, we will consider three test problems, as described in Table 4.1. Each of the problem domains will consist of a single fissile region padded on each side with a 5 mean-free-path reflector as shown in Figure 4.1. These tests have been modified from the sample problems described in [20, 17].

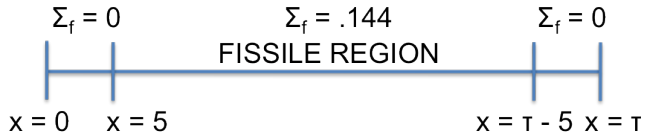

Figure 4.1: Fissile region surrounded by two 5 m.f.p. reflectors.

Table 4.1: One-Dimensional, One-Group, Slab Geometry k-Eigenvalue Test Problems

Test	1	2	3
Σ_s	.856	.856	.856
Σ_t	1	1	1
$\nu\Sigma_f$	.144	.144	.144
τ	110	60	35
N_x	5500	3000	1750

We will consider a total of 5 methods for solving these problems: power iteration for a baseline comparison, NDA-PI with and without the Wielandt Shift, NDA-NCA in which a Krylov method is used to compute the Newton step (NDA-NCA(K-PI)) and NDA-NCA in which a direct method is used to compute the Newton step (NDA-NCA(D)). For each test, we will use 40 angles in the angular quadrature. Both the inner and outer iteration will be converged to a tolerance of 10^{-12}. The convergence results for these test problems are displayed in Table 4.2. For NDA-PI and NDA-NCA, the first number refers to the number of transport sweeps required and the second number refers to the number of diffusion solves that take place.

It is clear from Table 4.2 that both NDA-PI and NDA-NCA provide major improvements over Power Iteration. It should be clear from these simple tests that standard power iteration is not a feasible method for solving the k-eigenvalue problem. Even for Test 3, which is relatively

Table 4.2: One-Dimensional, One-Group, Slab Geometry k-Eigenvalue Test Problems Convergence Results

Method	Test 1		Test 2		Test 3	
Power Iteration	52700	0	16780	0	5571	0
NDA-PI	10	3594	11	1547	12	517
NDA-PI(WS)	10	89	11	22	12	34
NDA-NCA(K-PI)	14	144	12	95	12	85
NDA-NCA(D)	10	20	11	22	12	24

simple, it takes on the order of 5000 transport sweeps to compute the eigenvalue. Compare this to the 10-12 transport sweeps required for to compute the eigenvalue using NDA-PI or NDA-NCA.

At this point it is reminding the reader that the number of diffusion solves does begin to effect the computational time required to compute the eigenvalue. While transport sweeps are significantly more expensive than diffusion solves, Table 4.2 demonstrates that thousands of diffusion solves may be required to compute the eigenvalue if the low-order eigenvalue problem is not accelerated. For each of these test problems, the table shows how effective the Wielandt shift can be when applied to the Low-Order eigenvalue problem.

Furthermore, we can see that NDA-NCA(D) outperforms each of the other methods considerably. The direct matrix inversion improves the algorithm in two distinct ways. First of all, it is far cheaper in terms of diffusion solves per outer iteration. The direct matrix inversion costs roughly the same as two preconditioner applications within the Krylov iteration. Secondly, the Krylov method only attempts to find a step w such that

$$\|F'(\phi)w + F(\phi)\| \leq \eta\|F(\phi)\|, \tag{4.32}$$

where η is generally chosen equal to .01. The direct matrix inversion, however, solves the linear system to machine tolerance. In essence, the direct matrix inversion computes a more accurate Newton step at a lower cost. This more accurate Newton step may lead to fewer outer iterations, as is demonstrated for Tests 1 and 2 in Table 4.2.

4.3.2 2D Test Problems and Results

In this section we'll consider three 2-D, 2 group multi-material eigenvalue problems. We'll display results for each of these problems individually in order to give a full breakdown of domain and material properties.

For each test we'll terminate on eigenvalue residuals less than 10^{-7}. As an initial iterate for

each method we'll use power iteration to solve the diffusion eigenvalue problem to a tolerance of 10^{-3}. This provides a good approximation to the eigenvalue and eigenvector and helps ensure that we converge to the dominant eigenvalue. It is important to realize that computing this diffusion eigenvalue is done only to proved a strong initial iterate. Solving the diffusion eigenvalue problem (Equation 4.23) to tighter tolerance makes no significant difference in the amount of iterations to convergence. For each of the following tests we will utilize an S_{10} angular quadrature.

Test 1

The first test is a 2-group, three material problem (TMR) as described in [13]. We display the material configuration in Figure 4.2 and list the material and domain properties in Tables 4.3 and 4.4.

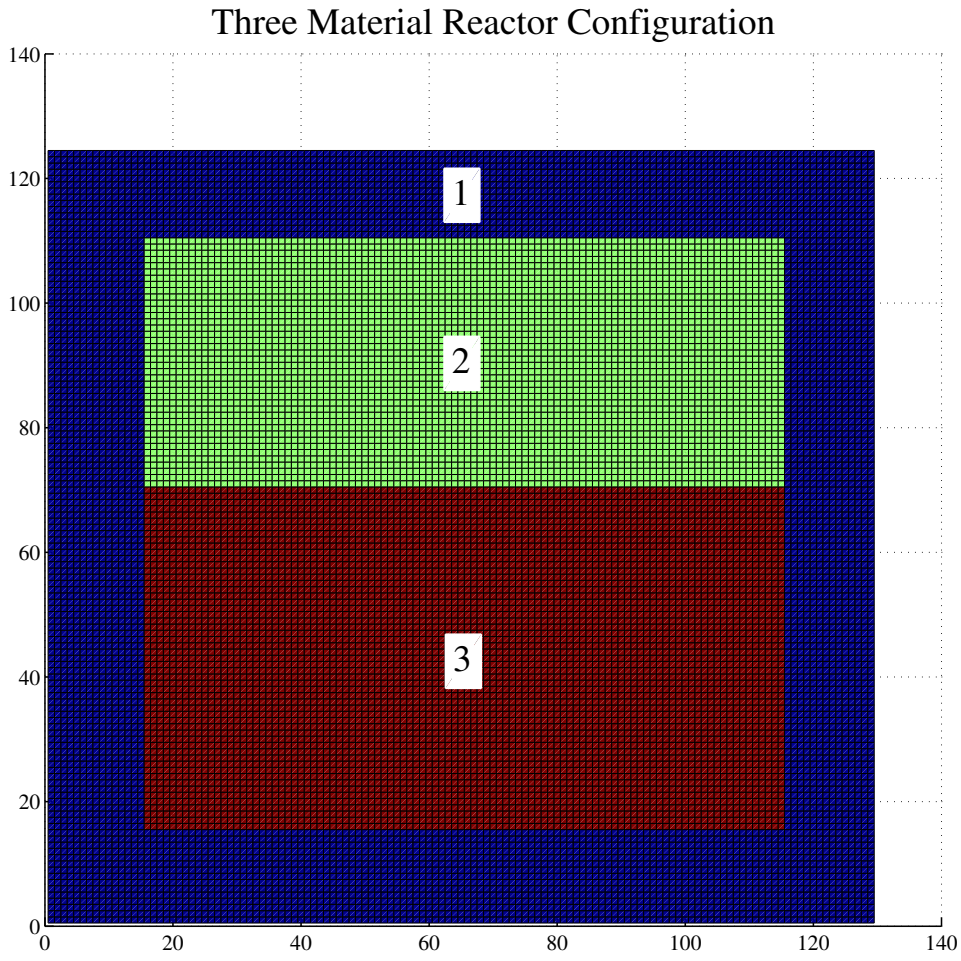

Figure 4.2: Three material reactor layout with regions labeled 1 - 3.

Table 4.3: 2-D Eigenvalue Test 1 Material Parameters

Material	$\Sigma_{t,1}$	$\Sigma_{t,2}$	$\Sigma_s^{1\to1}$	$\Sigma_s^{1\to2}$	$\Sigma_s^{2\to2}$	$\nu\Sigma_{f,1}$	$\Sigma_{f,2}$
1	0.2950	2.0080	0.2453	0.0493	1.9880	0	0
2	0.2631	0.9416	0.2269	0.0241	0.8206	0.0085	0.1851
3	0.2604	0.8333	0.2344	0.0160	0.7333	0.0060	0.1500

Table 4.4: 2-D Eigenvalue Test 1 Domain Parameters

τ_x	τ_y	N_x	N_y	χ_1	χ_2	Boundaries
125	130	125	130	1	0	All Vacuum

Test 1: Results

Table 4.5 demonstrates the convergence results for power iteration and the three NDA based acceleration algorithms. As we can see, for this problem power iteration converges quite slowly. Each of the NDA based algorithms converges in 10 outer iterations (10 transport sweeps). NDA-PI requires a significant number of low-order diffusion solves. Though each of these solves is less expensive than a transport sweep, the nearly 300 low-order solves do take a considerable amount of time. NDA-NCA(K) cuts the number of diffusion solves by a factor of roughly 4. NDA-NCA(D) improves on this number considerably. In Figure 4.3, we plot the eigenvector corresponding to the dominant eigenvalue.

Table 4.5: 2-D Eigenvalue Test 1 Results

Method	Transport Sweeps	LO Function Evaluations	Diffusion Solves
Power Iteration	5600	0	0
NDA-PI	10	0	299
NDA-NCA(K)	10	59	69
NDA-NCA(D)	10	10	20

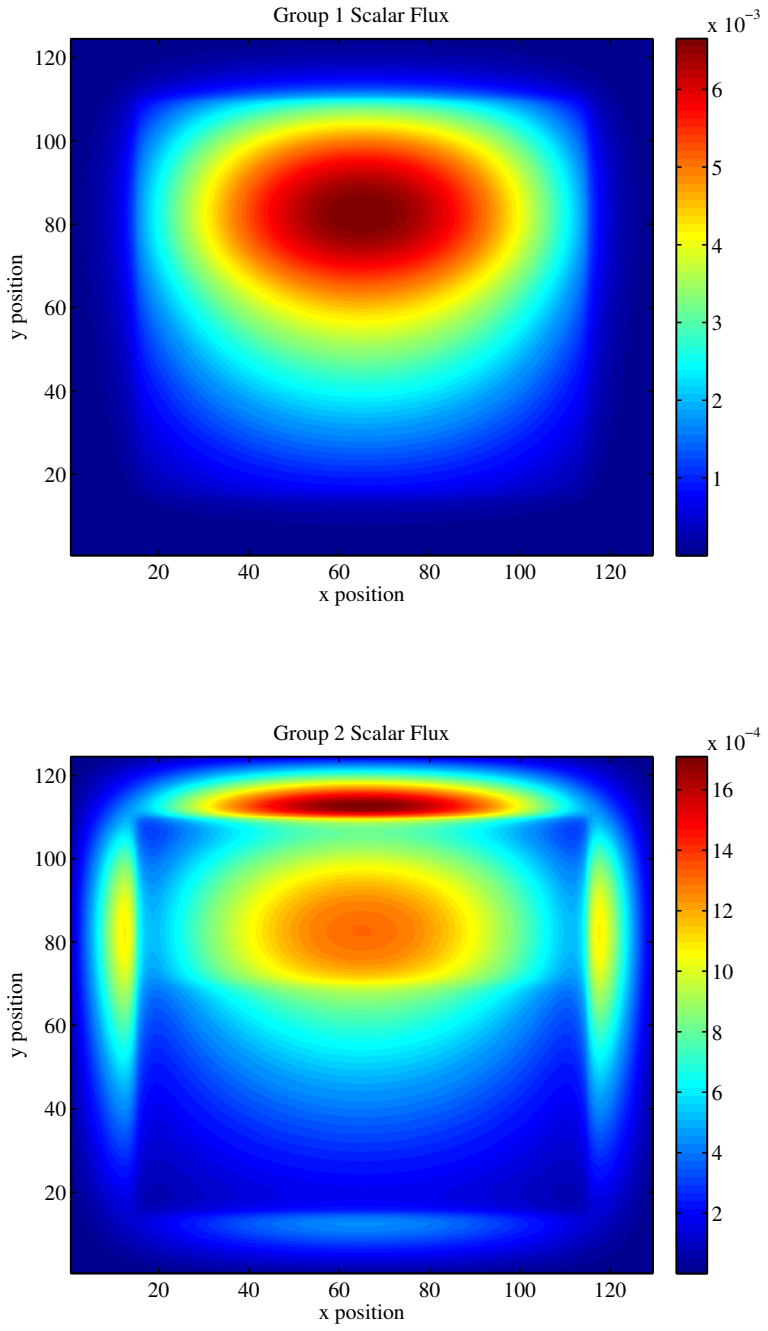

Figure 4.3: Plot of eigenvector corresponding to dominant eigenvalue for TMR problem.

Test 2: Zion Reactor

The second test is a 2-group, five material problem as described in [28, 1]. We display the material configuration in Figure 4.4 and list the material and domain properties in Tables 4.6 and 4.7.

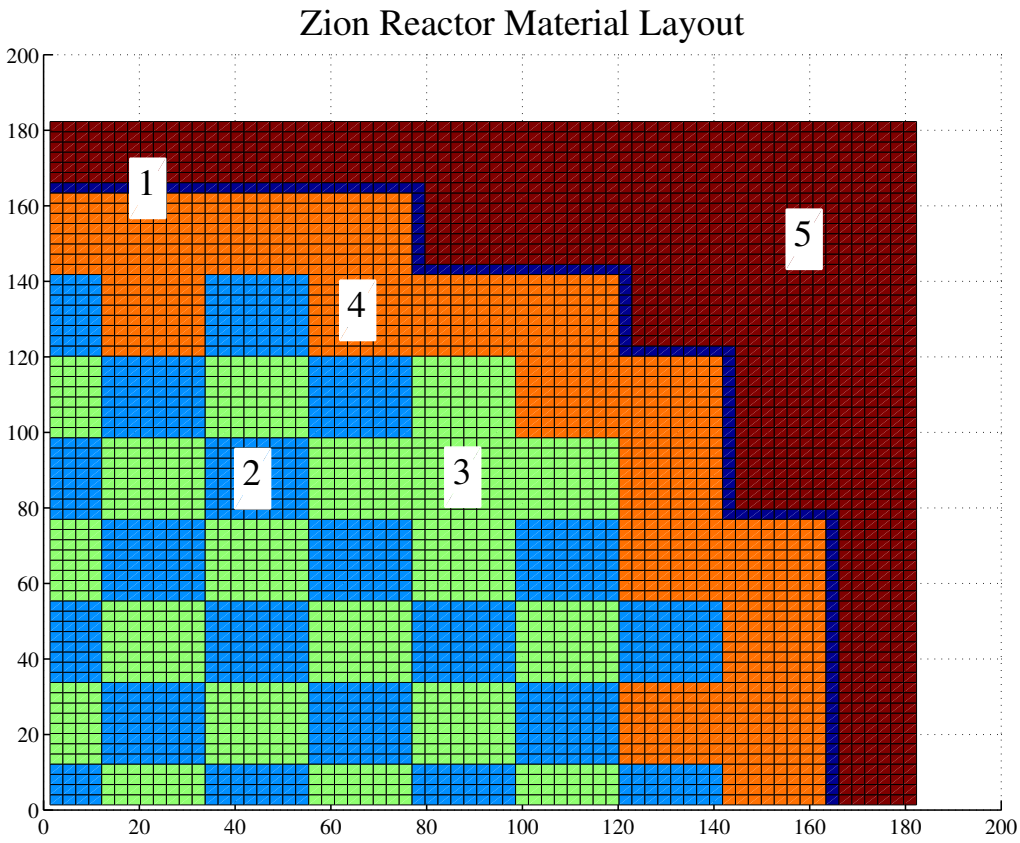

Figure 4.4: Zion reactor material layout with regions labeled 1 - 5.

Test 2: Results

The convergence results for Test 2 are given in Table 4.8. Much like Test 1, the convergence rate of power iteration very slow. Each of the NDA based methods converge in exactly 6 transport sweeps. This is represents over a 1400 times reduction in the number of transport sweeps

Table 4.6: 2-D Eigenvalue Test 2 Material Parameters

Material	$\Sigma_{t,1}$	$\Sigma_{t,2}$	$\Sigma_s^{1\to1}$	$\Sigma_s^{1\to2}$	$\Sigma_s^{2\to2}$	$\nu\Sigma_{f,1}$	$\Sigma_{f,2}$
1	0.3564	0.9936	0.3232	0.0000	0.8476	0	0
2	0.2351	0.8928	0.2093	0.0174	0.8261	0.0054	0.1043
3	0.2349	0.8920	0.2091	0.0169	0.8159	0.0060	0.1247
4	0.2337	0.8907	0.2083	0.0166	0.8071	0.0065	0.1412
5	0.2290	1.1497	0.1995	0.0290	1.1402	0.0005	0.0095

Table 4.7: 2-D Eigenvalue Test 2 Domain Parameters

τ_x	τ_y	N_x	N_y	χ_1	χ_2	Boundaries
183.67	183.67	136	136	1	0	Reflective Corner

required to compute the eigenvalue to the same accuracy. NDA-NCA provides a reduction in the number of low-order diffusion solves. Again, we see the dramatic increase in efficiency by using a direct inverse of the Jacobian when computing the Newton step. A plot of the dominant eigenvector can be found in Figure 4.5.

Table 4.8: 2-D Eigenvalue Test 2 Results

Method	Transport Sweeps	LO Function Evaluations	Diffusion Solves
Power Iteration	8750	0	0
NDA-PI	6	0	247
NDA-NCA(K)	6	50	44
NDA-NCA(D)	6	6	12

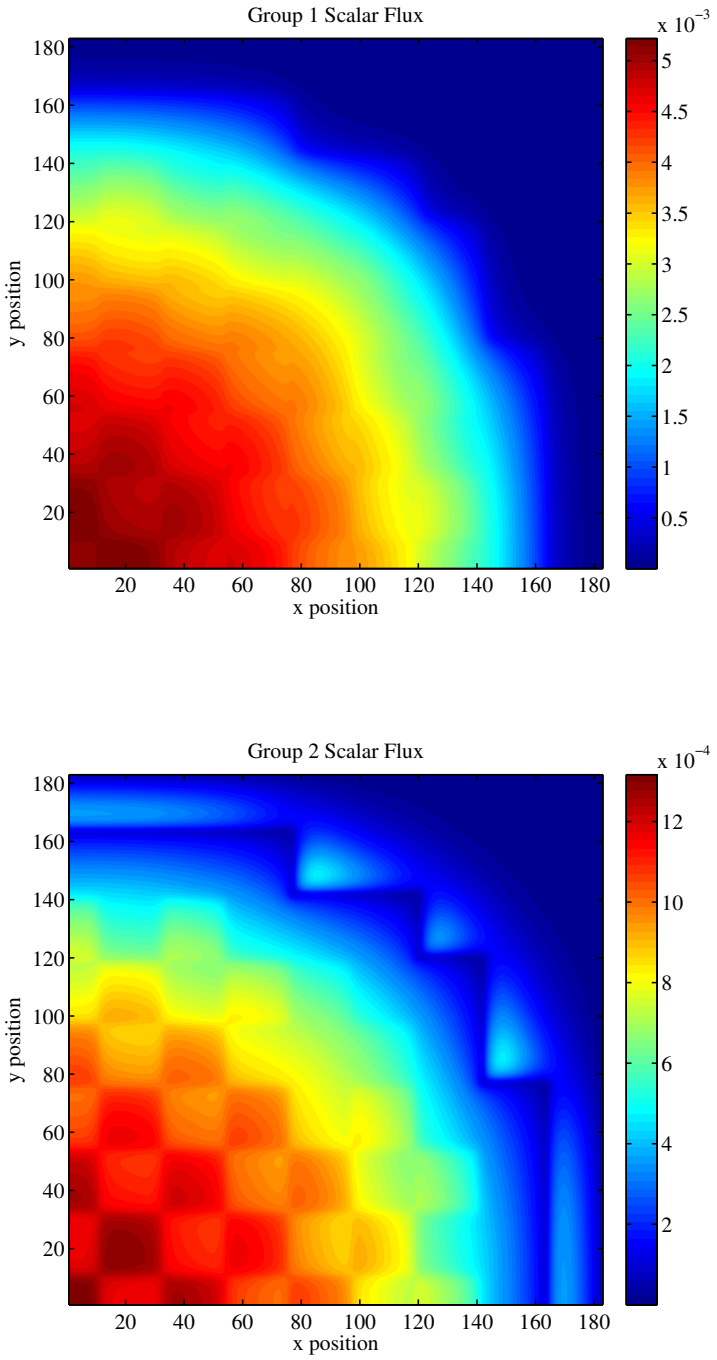

Figure 4.5: Plot of eigenvector corresponding to dominant eigenvalue for Zion reactor problem.

Test 3: LRA-BWR Reactor

The final 2-D test is a 2-group, five material problem as described in [28, 23, 1]. We display the material configuration in Figure 4.6 and list the material and domain properties in Tables 4.9 and 4.10.

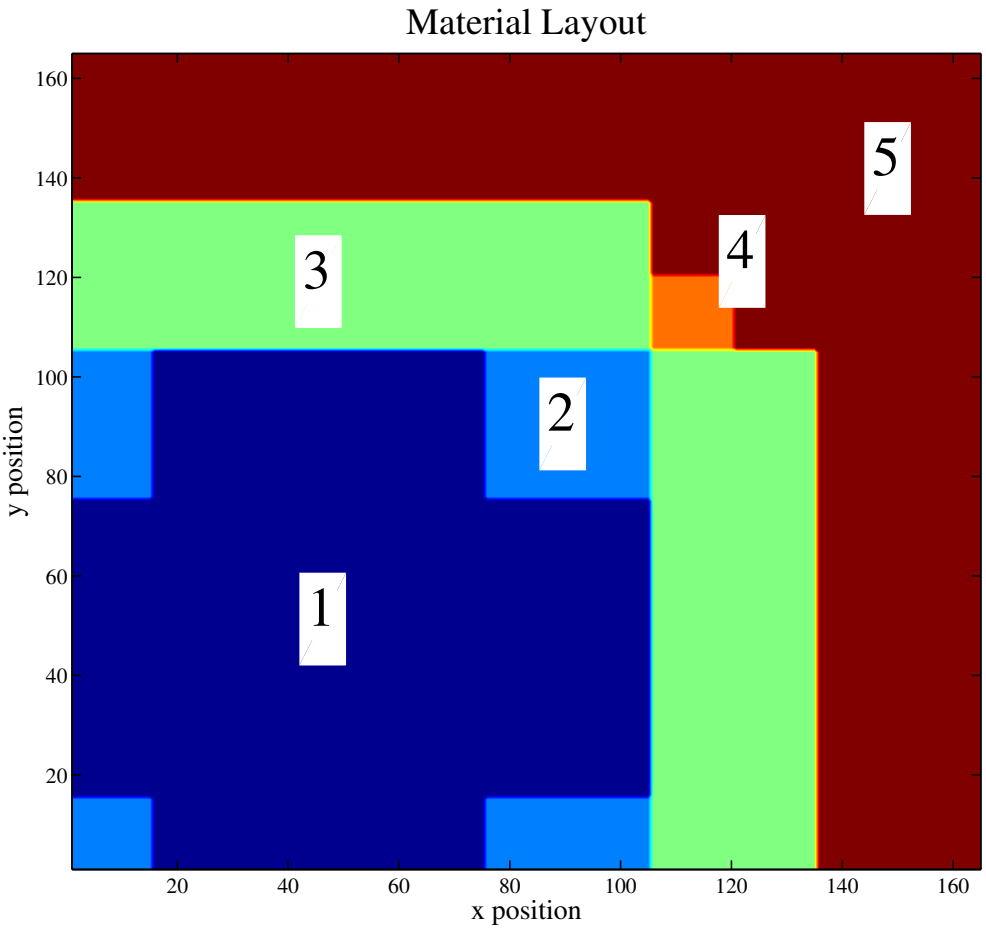

Figure 4.6: LRA-BWR material layout with regions labeled 1 - 5.

Table 4.9: 2-D Eigenvalue Test 3 Material Parameters

Material	$\Sigma_{t,1}$	$\Sigma_{t,2}$	$\Sigma_s^{1\to1}$	$\Sigma_s^{1\to2}$	$\Sigma_s^{2\to2}$	$\Sigma_{f,1}$	$\Sigma_{f,2}$
1	0.2656	1.5798	0.2321	0.0253	1.4795	0.0019	0.0449
2	0.2623	1.7525	0.2280	0.0277	1.6821	0.0019	0.0357
3	0.2648	1.5941	0.2306	0.0262	1.5107	0.0019	0.0420
4	0.2648	1.5941	0.2306	0.0262	1.5208	0.0019	0.0420
5	0.2652	2.0938	0.2170	0.0475	2.0747	0.0000	0.0000

Table 4.10: 2-D Eigenvalue Test 3 Domain Parameters

τ_x	τ_y	N_x	N_y	χ_1	χ_2	ν	Boundaries
165	165	165	165	1	0	2.43	Reflective Corner

Test 3: Results

The convergence results for the final test can be found in Table 4.11. For the LRA-BWR test problem, we find that power iteration requires over 15000 iterations. Both NDA-PI and NDA-NCA(D) require only 13 transport sweeps to converge to the same tolerance. NDA-NCA(K) requires an extra transport sweep, however this still represents over a 1000 times decrease in transport sweeps. Much like the first two tests, NDA-PI requires a significant number of low-order diffusion solves. While NDA-NCA(K) requires an extra transport sweep, the decrease in the amount of low-order work makes it more efficient than NDA-PI. Again, NDA-NCA(D) is the most efficient of the NDA-based algorithms. Figure 4.7 displays the dominant eigenvector for the LRA-BWR test problem.

Table 4.11: 2-D Eigenvalue Test 3 Results

Method	Transport Sweeps	LO Function Evaluations	Diffusion Solves
Power Iteration	>15000	0	0
NDA-PI	13	0	443
NDA-NCA(K)	14	98	84
NDA-NCA(D)	13	13	26

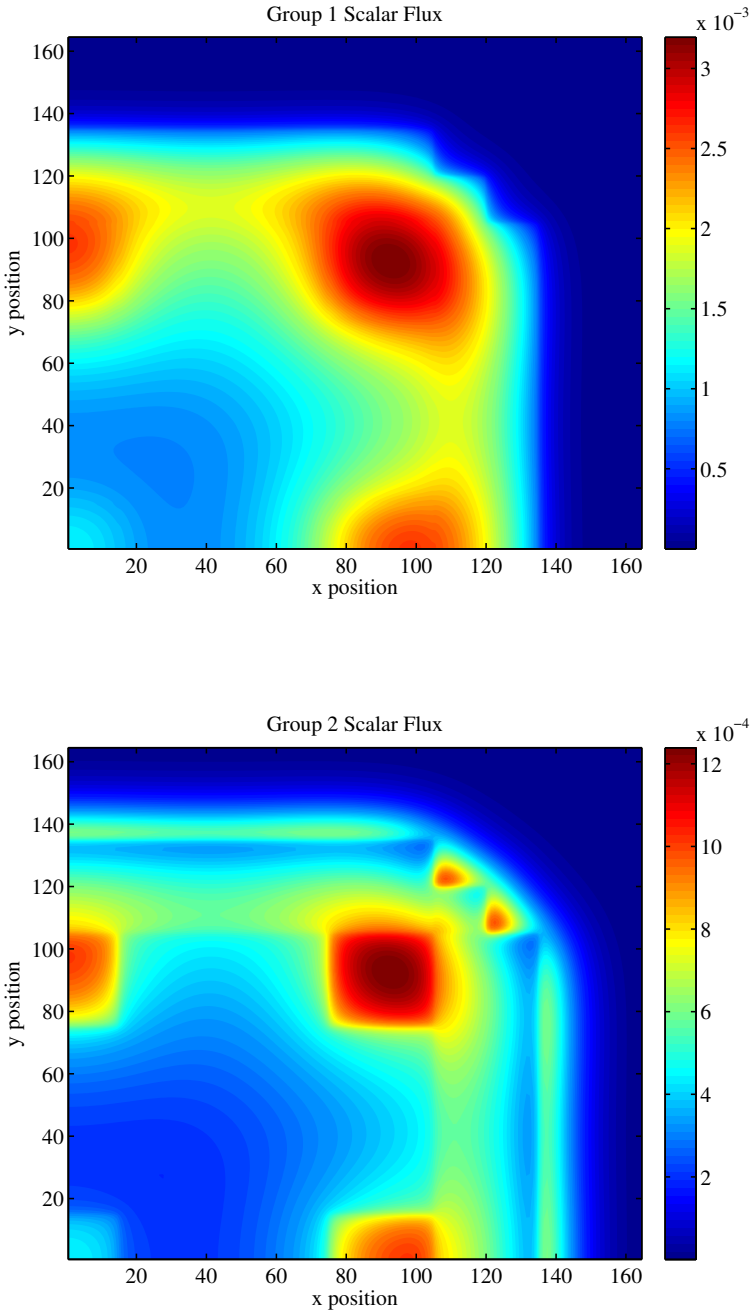

Figure 4.7: Plot of eigenvector corresponding to dominant eigenvalue for LRA-BWR problem.

4.4 Conclusions

In the preceding section we were able to demonstrate for a series of 1 and 2-D test problems that in every case examined, power iteration converges very slowly. For the easiest 1-D problem, power iteration still required over 5000 transport sweeps to converge to a relative tolerance of 10^{-12}. For the most difficult 2-D problem we examined, power iteration needed over 15000 transport sweeps to converge to 10^{-7}. It should be clear from this spectrum of test problems that unaccelerated power iteration converges unacceptably slow.

For each of these test problems, the three NDA-based methods we considered provide a remarkable speed-up over power iteration. These methods move much of the iterative work from the high-order transport equation to the low-order diffusion equation. Recall that these low-order iterations occur for a small phase space, one in which we've removed the angular dependence we see in the high-order angular flux.

For all six tests, at the very worst the NDA accelerated eigenvalue methods converge in under 15 iterations. This represents a 400-1200 times reduction in transport sweeps. We've demonstrated that these NDA-based methods now require low-order function evaluations and sparse, banded matrix solves at each iteration. In every case, NDA-PI requires the most low-order matrix solves. This number is reduced significantly by NDA-NCA(K) and reduced even further by NDA-NCA(D). NDA-NCA(D) is the most efficient algorithm for each of the 1-D and 2-D test problems examined.

Chapter 5

Stochastic Methods for Neutron Transport

In the previous chapters, we've focused primarily on determining the neutron scalar and angular flux by solving partial differential equations, namely the neutron transport equation and its angular moments. As we've mentioned, the neutron transport equation can be quite expensive to solve, especially when the material configurations are complex and we'd like to determine the flux over several energy groups or when using a large angular quadrature set. Furthermore, in many cases it may be cost prohibitive (in terms of memory or computational time) to use a dense enough angular quadrature set or spatial grid to capture all the physics required to get a valid picture of the physical state of a nuclear reactor [21].

In order to deal with these issues, many nuclear engineers choose to solve the neutron transport equation using stochastic methods. These methods allow for a continuous (or nearly continuous) treatment of space, angle and energy. This frees us of discretization errors and may allow us to capture the physics more exactly than we could while employing S_n methods. For example, suppose we discretize in space using rectangular grid cells. In this case, we are required to treat some regions of a reactor, such as cylindrical fuel pins, inexactly. Monte Carlo methods allow us to treat these spatial features more precisely.

5.1 Introduction to Monte Carlo

This chapter will act as an introduction to Monte Carlo methods for solving the neutron transport equation. Monte Carlo methods attempt to realize probablistic events using the sampling of random numbers [21]. For example, suppose we'd like to simulate the flipping of a fair coin and we wish to determine the result of a flip - either heads or tails. We can simulate the outcome of the event by selecting a uniformly distributed random number $\xi \in [0, 1)$. We will

assign the outcome "heads" whenever $\xi \in [0, .5)$ and "tails" otherwise.

Monte Carlo methods also give us the ability to simulate more complicated events. Suppose we have a probability distribution function, $f(x)$, which describes the probabilty of a random variable taking certain values. The probability that x falls between two values, a and b, can be determined by

$$\Pr[a \leq x \leq b] = \int_a^b f(x)dx.$$

Furthermore, we can describe the probability that x is less than some value a by

$$F(a) = \Pr[x \leq a] = \int_{-\infty}^{a} f(x)\ dx,$$

where $F(x)$ is called the *cumulative distribution function* or CDF.

Now, suppose we wish to use random number generation to pick x's according to the probability distribution function, $f(x)$, with associated cumulative distribution function, $F(x)$. We begin by taking a uniformly distributed random number $\xi \in [0, 1)$. Then, we compute the value of the random variable x^* by finding x such that

$$F(x) = \int_{-\infty}^{x} f(x)dx = \xi$$

or simply define x^* as

$$x^* = F^{-1}(\xi).$$

Using these simple ideas, we can simulate the entire life of a neutron using a sequence of a random numbers.

In practice, during numerical simulations we store the probability density function as a vector, $\vec{f}$, where each component of the vector corresponds to a measure of the scalar flux (or current, etc) in a given spatial cell. In this case, we compute the CDF using a summation

$$\vec{F}_i = \frac{\sum_{j=1}^{i} f_j}{\sum_{j=1}^{N} f_j}.$$

Now, when we pick a random number ξ, we must interpolate the vector $\vec{F}$ to find a value of x corresponding to ξ. When we choose to use a large number of particles, it may be more efficient to build a look-up table to aid in the computation of x.

5.2 Monte Carlo for Neutron Transport

If we were to track the entire life of a neutron we'd need to know the following information:

1. Where was the particle born?
2. What direction did the particle travel?
3. How far did the particle travel in that direction before either leaving the system or undergoing a collision?
4. If the particle collides, does the collision interaction result in scattering, absorption, fission?
5. If the particle scatters, which direction does the particle travel after scattering, what is its energy after the collision, and how far does it travel?
6. If the collision results in a fission event, how many fission neutrons are created, which direction do these neutrons travel, with what energies are the fission neutrons born, etc.?

If we can determine probabilities for these events, either experimentally or by deriving these probabilities based upon the underlying physics, we can use the Monte Carlo method to simulate the lives of thousands, millions, or billions of particles. As we increase the total number of particles in our simulation, we can get closer and closer to determining the true values of the neutron flux and current. As the number of particles, N, increases, the variance in the simulation decreases like $\mathcal{O}(\frac{1}{\sqrt{N}})$ [21].

5.2.1 Determining the Mono-energetic Neutron Flux in 1-D Homogeneous Slab Geometry with a Fixed Source

Suppose we wish to determine the neutron scalar flux in a homogeneous slab where all neutrons live within a single energy group and are produced by a constant fixed source. Furthermore, let us assume that scattering is isotropic. We'll define the following cross-sections:

$$\begin{aligned} \Sigma_t(x) &= \Sigma_t \\ \Sigma_a(x) &= \Sigma_a \\ \Sigma_s(x) &= \Sigma_s \\ \Sigma_f(x) &= 0 \end{aligned}$$

Furthermore, define the total system length, τ, and some fixed source distribution $S(x)$. We wish to determine the scalar flux, $\phi(x)$, and the neutron current, $J(x)$.

We'll begin by defining a probability distribution function that describes the distribution of particles generated by the right-hand side source term (scattering plus fixed sources). $S(x)$ is not necessarily a probability distribution function because there is no guarantee that

$$\int_0^\tau S(x)\ dx = 1$$

and, in fact, this equality will not hold in most cases. To deal with this, we define the probability distribution function, $Q(x)$, by

$$Q(x) = \frac{S(x)}{\int_0^\tau S(x)\ dx}.$$

Now, the integral of $Q(x)$ over the entire domain must be one. We will sample from this source distribution instead of $S(x)$ and then weight particles according to the total number of particles in the simulation and the source strength.

Let us consider the life of the i^{th} particle in the simulation. The first thing we need is an initial location. To do so, we pick a uniformly distributed random number ξ_1 and we choose our initial location x_0 defined by

$$\xi_1 = \int_0^{x_0} Q(x)dx.$$

In this section, we will assume all random numbers are uniformly distributed over the interval $[0, 1)$. Once we have our initial location, x_0, we need an initial direction. Instead of sampling the initial direction explicitly we will sample the direction cosine, $\mu = \cos(\theta)$. Since μ is uniformly distributed over the interval $[-1, 1]$, we pick a second random number ξ_2 and define our initial direction cosine by $\mu = 2\xi_2 - 1$.

Now that we have an initial location and direction for the particle, we need to determine how far the particle travels before undergoing its first collision. The probability of traveling a distance of z mean-free-paths in a material without colliding is given by

$$f(z) = e^{-z}.$$

Therefore, the cumulative distribution function is given by

$$F(z) = \int_0^z e^{-z'} dz'.$$

We can determine the distance the particle travels by generating a third random number ξ_3

and solving for z

$$\xi_3 = \int_0^z e^{-z'} dz'.$$

A bit of calculus reveals

$$z = -\ln(1 - \xi_3). \tag{5.1}$$

However, since ξ_3 is uniformly distributed in $[0, 1]$, $1 - \xi_3$ is also uniformly distributed in $[0, 1]$. Therefore, Formula 5.1 is equivalent to

$$z = -\ln(\xi_3).$$

This simplification may seem trivial, but can result in a significant time reduction when we consider that this computation may take place billions of times. In terms of distance, the particle will travel

$$x_{stream} = \frac{z}{\Sigma_t}$$

centimeters.

At this point, we have the particle's initial location, x_0, initial direction, μ, and the distance traveled before the first collision, x_{stream}. The location of the first collision is given by $x_{collision} = x_0 + \mu \cdot x_{stream}$. Before the simulation can continue, we must determine if $x_{collision}$ is located within the slab. If $x_{collision}$ lies within the slab, we set $x_1 = x_{collision}$. If $x_{collision}$ lies outside the domain, we need to check to determine whether it crossed a reflective or vacuum boundary. If the boundary was reflective, the particle collision location is reflected back into the domain, otherwise the life of the particle is over.

If the particle remains within the slab at the point of the first collision, we must determine whether the particle undergoes a scattering or an absorption reaction. We generate a new random number ξ_4 and say that the particle scatters if $\xi_4 \leq \frac{\Sigma_s}{\Sigma_t}$ and the particle is absorbed if $\xi_4 > \frac{\Sigma_s}{\Sigma_t}$ If the particle is absorbed, the particle history is terminated. If the particle lives on, this process is repeated where a new direction is computed, we determine a distance for the particle to travel and then test again for streaming out of the slab, scattering, or absorption until the particle history terminates. Once all of the particle histories have been terminated and recorded, we tally the results and report the scalar flux and current. In practice, we tally these quantities on the fly to avoid storing all of the particle histories.

5.2.2 Tallying Quantities of Interest

When tallying quantities like the scalar flux and current, our choice as to how to tally is influenced by one basic consideration - where do we need our data? In some applications, we may need values of the scalar flux at cell edges, while in other applications it may be most helpful to have cell-averaged quantities. Furthermore, it may be necessary to compute the cell-averaged scalar flux, but cell edge values of the current. For this reason, we develop methods for tallying each of the moments at cell edges and as cell-averaged quantities.

Before continuing, let us introduce some notation and definitions that will allow us to tally these quantities. Let n_j denote the j^{th} particle in the simulation. Each neutron in the simulation takes a series of flights $k = 1, 2, ..., N_j$, defined as the flights between collisions. That is, a neutron which undergoes three collisions before either leaving the medium or being absorbed takes four particle flights: one flight from birth to the first collision, a flight between the first and second collisions, a flight between the second and third collisions, and a flight after the third collision before leaving the medium or being absorbed.

Define the set $X_{i+\frac{1}{2}}$ as the set of all particle flight indices (j, k) such that the k^{th} flight of the j^{th} particle in the simulation crossed the cell face between cells i and $i+1$. Furthermore, let $w_{j,k}$ denote the weight of the j^{th} particle during the k^{th} flight. In this case, the term "weight" does not refer to a measure of the mass of the particle, but instead a factor attached to the particle that measures its contribution to certain tallies. In classic Monte Carlo simulations, $w_{j,k} = w_{j,0}$ for all k, where $w_{j,0}$ is the weight of a particle at birth and is generally constant for all particles or is proportional to the source strength at the birth location. When variance reduction techniques are in place, particle weights may vary based upon how many collisions a given particle has undergone or the initial particle flight direction [21, 5].

Furthermore, let $W_{Total} = \sum w_{j,0}$, that is, the sum of all initial particle weights. Since neutrons change direction after each interaction, we'll let $\mu_{j,k}$ denote the direction of travel of the j^{th} particle during its k^{th} flight.

Cell Edge (Face) Tallies

We begin by tallying the scalar flux at a cell face, that is some location $x_{i+\frac{1}{2}}$. The scalar flux can be interpreted as the total path-length traced by all particles per unit volume [21]. Therefore, we can realize the scalar flux ϕ at the location $x_{i+\frac{1}{2}}$ by

$$\phi_{i+\frac{1}{2}} = \frac{1}{W_{Total}} \left(\sum_{(j,k) \in X_{i+\frac{1}{2}}} \frac{w_{j,k}}{|\mu_{j,k}|} \right) \tag{5.2}$$

Recall that the current, J, is given by

$$J(x) = \int_{-1}^{1} \mu\psi(x,\mu)d\mu.$$

Therefore, the current can be interpreted as the total path-length traced by all neutrons at the location $x_{i+\frac{1}{2}}$ scaled by the direction of travel μ:

$$J_{i+\frac{1}{2}} = \frac{1}{W_{Total}} \left(\sum_{(j,k)\in X_{i+\frac{1}{2}}} w_{j,k} \frac{\mu_{j,k}}{|\mu_{j,k}|} \right) \tag{5.3}$$

Cell-Averaged Tallies

We can compute cell-averaged tallies is a similar way. This time, define the set Y_i as the set of all particle flight indices (j,k) such that the k^{th} flight of the j^{th} particle takes place (at least partially) within the i^{th} cell. Furthermore, define $\Delta x_{i,j,k}$ as the $x-$distance traveled within the i^{th} cell during the k^{th} flight of the j^{th} particle.

Now, we tally the three quantities of interest in a manner similar to that of the cell-edge tallies. The scalar flux and current are given by

$$\phi_i = \frac{1}{\Delta V_i W_{Total}} \left(\sum_{(j,k)\in Y_i} \frac{w_{j,k}\Delta x_{i,j,k}}{|\mu_{j,k}|} \right) \tag{5.4}$$

$$J_i = \frac{1}{\Delta V_i W_{Total}} \left(\sum_{(j,k)\in Y_i} \frac{w_{j,k}\Delta x_{i,j,k}\mu_{j,k}}{|\mu_{j,k}|} \right) \tag{5.5}$$

where ΔV_i refers to the volume, or width, of the ith cell.

Comparing the Cell-Edge and Cell-Averaged Tallies

As we've noted earlier, there are times in which we may need the scalar flux and current tallied at cell interfaces and other times where we may need these quantities tallied as cell averages. The question arrises, however, if neither tally method is preferred by the algorithm which we are currently employing, is one of the tally methods preferable to the other?

The answer to this question is complicated and may not have a definitive answer, but there are a couple of important notes which we should make. First of all, as the spatial mesh width, h, goes to zero and the number of particle histories, N, approaches infinity, the two tally methods will limit to the same solution. Unfortunately for us, we are never able to tally infinitely many particles, and for a finite number of particle histories, the two tally methods may yield slightly

different results. So, is one of these tally methods "better" than the other?

One can argue that the cell-average tally method provides better results because it offers a more "continuous" treatment of the tallies. For example, with the cell-edge tallies, we can only count the particles which actually cross a cell interface. If a particle is born in cell i and its history is terminated before leaving cell i, it will make no contribution whatsoever in the cell-edge tally system. However, in the cell-averaged tally method, every particle has at least some contribution. Whether a particle travels across several cells or some short distance within a single cell, every particle will influence the cell-average tally.

On the other hand, these tallies may consume the majority of the computational time in a Monte Carlo simulation. Therefore, whenever possible we should seek to only return either a cell-edge or cell-averaged tally, but not both. If there is need for cell-edge and cell-averaged quantities of either the flux or current , we may be able to get away with computing only a cell-edge tally and then taking an arithmetic average of two cell-edge values to find an approximate cell-averaged value. This averaging is far less computationally intensive than executing both tally methods.

In summary, most times the choice of whether to use the cell-average or cell-edge tally will be determined by the specific problem being solved and the algorithm used to solve the given problem. If the algorithm has no preference as to where these quantities are tallied, it may be preferable to exploit the more "continuous" method of tallying found in the cell-average tally.

5.3 Comparing Source Iteration and Monte Carlo

Recall from Chapter 2 the Source Iteration method for solving the steady-state 1-D monoenergetic transport equation in slab geometry. We must choose some initial iterate $\phi^{(0)}(x)$ and in this case, let us choose $\phi^{(0)}(x) \equiv 0$. Then we iterate as follows:

$$\mu \frac{\partial \psi^{(i)}}{\partial x} + \Sigma_t \psi^{(i)} = \frac{1}{2}\left(\Sigma_s \phi^{(i)} + q\right) \tag{5.6}$$

$$\phi^{(i+1)} = \int_{-1}^{1} \psi^{(i+1)} d\mu \tag{5.7}$$

In this case, $\psi^{(i)}$ represents the angular flux due to particles that have scattered at most i times [2].

It was our intention in Chapter 2 to demonstrate exactly how Source Iteration is employed using deterministic methods. In this section, we'll describe a way in which Source Iteration can be utilized in conjunction with Monte Carlo. With deterministic methods, all we needed was a

function evaluation that took as input $\phi^{(i)}$ and returned $\phi^{(i+1)}$:

$$\phi^{(i+1)} = G_{DET}(\phi^{(i)})$$

where the action of G is often referred to as a "transport sweep". The subscript "DET" refers to the deterministic nature of the function. It turns out, however, that we can easily define a function G_{MC} which performs the same action as G_{DET} using the Monte Carlo method.

Essentially, this function must take in the flux due to particles which have scattered at most i times and return the flux due to particles which have scattered at most $i+1$ times. We accomplish this by allowing Monte Carlo to solve a new problem,

$$\mu \frac{\partial \psi^{(i)}}{\partial x} + \Sigma_t \psi^{(i)} \quad = \quad S^{(i)}(x) \tag{5.8}$$

where

$$S^{(i)}(x) = \frac{1}{2}\left(\Sigma_s \int_{-1}^{1} \psi^{(i-1)}(x,\mu)d\mu + q(x)\right).$$

In this new problem, $S^{(i)}$ is treated as a constant, fixed source, so the Monte Carlo method is asked to solve a *scattering-free problem*. In essence, every particle dies at the location of its first collision. This action allows us to compute the new flux due to all particles that have scattered at most $i+1$ times. Therefore, we can define a new function G_{MC} which takes a flux as its input and allows each of these particles to scatter one more time before ending the particle history.

We will put this idea to use in Chapter 6, Hybrid Methods for Neutron Transport. In theory, we can compute a transport sweep by using Monte Carlo for any of the algorithms previously mentioned. As the number of particle histories approaches infinity and the number of spatial cells and angles go to infinity, both the deterministic and Monte Carlo transport sweeps should yield the same results. However, for finitely many particle histories, these results will differ and the stochastic nature of the Monte Carlo evaluation of the transport sweep may cause drastic issues.

5.3.1 Monte Carlo Simulation Notation

Throughout the remainder of this thesis there will be times where it will be important to note how the scattering source is being treated within the Monte Carlo simulation. In general, when we ask Monte Carlo to simulate scattering, we'll move the scattering source to the left-hand

side of the transport equation,

$$\mu\frac{\partial\psi}{\partial x}+\Sigma_t\psi-\frac{1}{2}\int_{-1}^{1}\Sigma_s\psi\ d\mu=\frac{1}{2}q \tag{5.9}$$

With this notation it should be clear that Monte Carlo seeks to simulate all of the terms on the left-hand side of the equation.

In contrast, when we use Monte Carlo simulations to execute a transport sweep, we'll utilize the standard formulation of the transport equation,

$$\mu\frac{\partial\psi}{\partial x}+\Sigma_t\psi=\frac{1}{2}\left(\Sigma_s\phi+q\right), \tag{5.10}$$

which we are used to from the first few chapters. In this case, since the scattering source is considered an input to the simulation it has been moved to the right-hand side of the equation. Now, the Monte Carlo algorithm will simulate only leakage and absorption.

We can also combine the previous two ideas and allow the Monte Carlo algorithm to simulate some fraction of the scattering source. In this case we write

$$\mu\frac{\partial\psi}{\partial x}+\Sigma_t\psi-\frac{\alpha}{2}\int_{-1}^{1}\Sigma_s\psi\ d\mu=\frac{1-\alpha}{2}\Sigma_s\phi+\frac{1}{2}q, \tag{5.11}$$

where $\alpha\in[0,1]$. In the case of $\alpha=0$, Equation 5.11 reduces to Equation 5.10. In the case of $\alpha=1$, Equation 5.11 reduces to Equation 5.9. In general, Monte Carlo transport sweeps may be noisier than a full Monte Carlo simulation as there are generally fewer particle flights contributing to the physical quantities which we tally. If this noise becomes a problem, we may be able to partially remedy this issue by simulating a fraction of the scattering source without accruing the cost of a full Monte Carlo simulation.

5.4 Continuous Energy Deposition Tallies

In Section 5.2.2, we discussed two methods of tallying, face, or cell-crossing, tallies and cell, or track-length, tallies. We will refer to these as *standard tallies* for the remainder of this thesis. It turns out that there is a very fundamental reason which these tallies are sub-optimal [10, 32].

Let us begin by taking the zeroth angular moment of Eq. 5.8. This yields the neutron balance equation,

$$\frac{dJ}{dx}+\Sigma_t\phi=S(x). \tag{5.12}$$

When we tally the scalar flux and current, ϕ and J, respectively, with conventional track-length and cell-crossing estimators, we find that Eq. 5.12 is not satisfied. In fact, if we define a residual

based on this balance equation,

$$R(x;\phi,J) = \frac{dJ}{dx} + \Sigma_t\phi - S(x) \tag{5.13}$$

we find that $||R|| \gg 0$ when standard tallies are used to compute ϕ and J [32]. As the number of particles per simulation increase, $\|R\|$ decreases, but for a reasonable number of particles, R can be substantial. Along with the noise we find in the Monte Carlo transport sweeps, this lack of balance can produce inaccurate results when trying to compute the consistency term, $\hat{D}$, which measures the discrepancy in the Fick's law approximation.

Now, we present a second tally which we will refer to as the *Continuous Energy Deposition tally* or *CED tally* [10]. This tally is especially suitable for NDA-based methods since the Monte Carlo simulation solves a problem with purely absorbing media. The purpose of the CED tally is to conserve the balance of neutrons which is required by the 0^{th} moment equation.

The CED tally takes place in two distinct phases:

1. A non-analog tally in which each particle is forced to travel a fixed mean-free-path distance in which we tally using exponentially decreasing weights.

2. A standard tally from the end-point of the non-analog tally using the remaining particle weight.

Our new Monte Carlo simulation takes place as follows:

1. A priori determine a number of mean-free-paths, λ, which each particle will travel in the non-analog phase of tallying. We choose λ directly, but one could also choose λ so that the non-analog phase terminates when a certain minimum weight has been reached. These are equivalent.

2. Randomly sample the direction cosine, μ, and determine an initial location for a particle as described in Section 5.2.1 .

3. March across the medium until the particle has traveled the a priori determined number of mean-free-paths, λ.

    - Compute the distance in terms of mean-free-paths traveled within cell i:

    $$\Delta\lambda_i = \frac{\Delta x_i \Sigma_t}{|\mu|}$$

    - Compute the weight for the track-length tally of the scalar flux:

    $$\Delta w_i \left(1 - e^{-\Delta\lambda_i}\right)$$

- Tally the scalar flux using the weight Δw_i using the track-length tally described in Section 5.2.2
- Compute the weight at the cell interface:

$$w_{i\pm1} = w_i e^{-\Delta\lambda_j}.$$

- Tally the current using the weight $w_{i\pm1}$ using the face-tally described in Section 5.2.2.
- Repeat for the next cell.

4. Begin an analog tally from the new particle location using $w = w_0 e^{-\lambda}$. We use the same direction cosine, μ, as was used in the non-analog phase of the tally.

The CED tally satisfies the balance equation to the point of machine roundoff. We can see this in Figure 5.1. Notice the difference in scales on the y-axis.

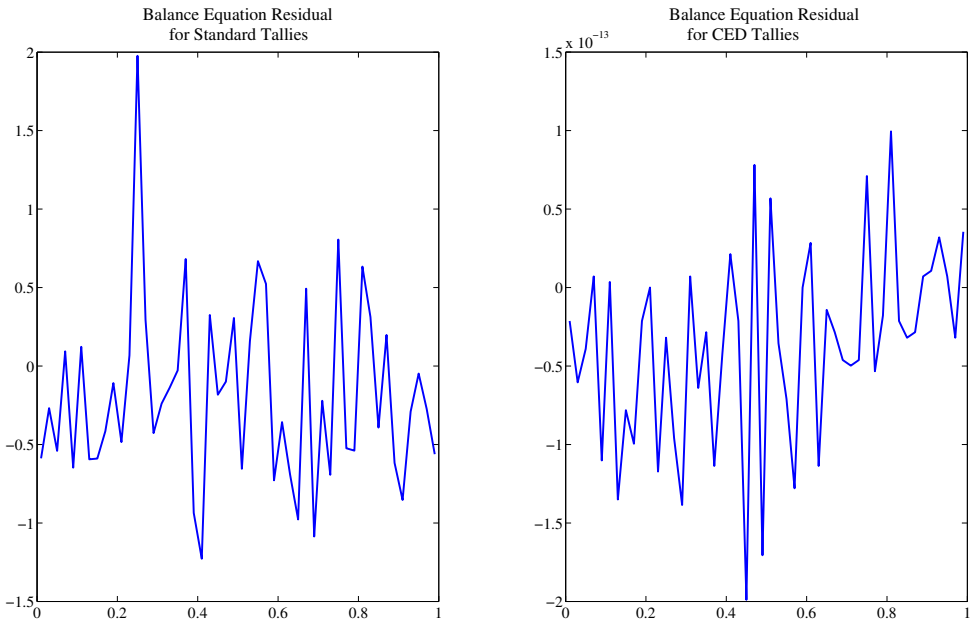

Figure 5.1: On the left we see the standard tally does not preserve balance, whereas on the right, the CED tally does an excellent job preserving balance using the same number of particles per simulation.

In Chapter 6, we'll compare results using standard tallying methods and CED tallies. We'll see that for low particle counts, the CED tally provides significantly smoother results, however as the particle count increases, the differences become smaller.

5.5 2-D Monte Carlo

In multiple dimensions, not much changes when executing a Monte Carlo simulation to determine the scalar flux and currents. The overall structure of the Monte Carlo code does not change at all. In general, we still must sample for a particle's starting location and weight, w_i, its initial direction and energy, and the number of mean-free-paths it travels before undergoing its first interaction [21].

In two spatial dimensions, the particle location now has an x and y-position as well as an x and y component to the direction. In order to sample the direction of travel, we require two random numbers, ξ_1 and ξ_2. Now, the x and y components of the direction of travel, $\hat{x}$ and $\hat{y}$, respectively, are given by

$$\begin{aligned} \hat{x} &= \sin(\theta)\cos(\phi) \\ \hat{y} &= \sin(\theta)\sin(\phi) \end{aligned}$$

where

$$\begin{aligned} \mu = \cos(\theta) &= 2\xi_1 - 1 \\ \phi &= 2\pi\xi_2 \\ \sin(\theta) &= \sqrt{1-\mu^2}. \end{aligned}$$

We compute the distance a particle travels in the same way as we did in 1-D. We compute a single random number, ξ_3, and the number of mean-free-paths to be traveled is given by

$$mfp = -\ln(\xi_3).$$

As in 1-D, we can compute fluxes and currents at cell-centers or cell-faces. For our purposes, we'll compute cell-averaged values of the scalar flux and cell-face values of the $x-$ and y-currents. For either formulation, we need to compute the distance a particle will travel in the current cell before either undergoing a collision or leaving the cell. If the particle leaves the cell, we must determine which face the particle exited.

To do so, we first compute the x and y distances to the nearest cell interfaces in the direction

of travel,

$$\begin{aligned} x_{dist} &= |x_{position} - x_{interface}| \\ y_{dist} &= |y_{position} - y_{interface}|. \end{aligned}$$

We determine that we'll cross the cell boundary in the x direction first if we have

$$\frac{x_{dist}}{\hat{x}} < \frac{y_{dist}}{\hat{y}}$$

and we'll cross the cell boundary in the y direction first otherwise. At this point, we'll allow the particle to travel t mean-free paths, where t is given by

$$t = \min\left(\frac{x_{dist}}{\hat{x}}, \frac{y_{dist}}{\hat{y}}\right) \cdot \Sigma_t$$

Now, we can tally the quantities of interest. We accumulate the cell-averaged scalar flux via

$$\phi(x_{cell}, y_{cell}) = \phi(x_{cell}, y_{cell}) + \frac{w_i \cdot t \cdot \Sigma_t}{h_x \cdot h_y},$$

where h_x and h_y are the $x-$ and $y-$dimensions of the current cell. If the particle crosses the x cell boundary, we'll accumulate the current in the $x-$direction,

$$J_x(x_{bound}, y_{cell}) = J_x(x_{bound}, y_{cell}) + \frac{w_i \cdot \hat{x}}{|\hat{x}| \cdot h_y},$$

and if the particle crosses the y cell boundary, we'll accumulate the current in the $y-$direction,

$$J_y(x_{cell}, y_{bound}) = J_y(x_{cell}, y_{bound}) + \frac{w_i \cdot \hat{y}}{|\hat{y}| \cdot h_x}.$$

At this point, we've finished tallying for the flight through (x_{cell}, y_{cell}) and we continue to travel and tally cell by cell until the particle leaves the medium or encounters the location of its first collision.

5.6 Standard vs. CED Tallies in 2D

In this section we'll briefly look at the difference between standard and CED tallies. To do so, we'll generate particles that are uniformly distributed across a single cell in 2-D. We'll isotropically generate directions for each particles to travel and count the number of cells each particle crosses before it undergoes a collision. Each cell is 1 cm by 1 cm.

We'll compare the mean number of cells that each particle contributes to using both standard and CED tallies. In practice, if each particle does not cross at least a couple of cells, the Monte Carlo transport sweep is exceptionally noisy. The results from this study are portrayed in Table 5.1.

Table 5.1: Mean Number of Cells Crossed by Each Particle

Method	$\Sigma_t = 2$	$\Sigma_t = .3$
Standard	1.4976	4.3590
CED ($\lambda = 2$)	2.4897	11.0371
CED ($\lambda = 4$)	3.4821	17.7343

Clearly, the CED tallies allow each particle to stream across more cells than when standard tallies are employed. This is important because the current is only tallied when a particle cross a cell face. So, while a particle always contributes to the scalar flux, when the cross section is large, many particles may not contribute at all to the current. For example, in the above configuration with $\Sigma_t = 2$, 62% of the particle trajectories produced using standard tallies never left the originating cell. When CED tallies with $\lambda = 2$, only 10% of particles remain in the originating cell and this decreases to 3% when $\lambda = 4$. Therefore, on average, each particle contributes to more tallies for the scalar flux and current and produces less noisy results when the CED tallies are used. Even when $\Sigma_t = .3$, 18% of standard tally particles never leave the originating cell. Only .2% of CED tallies with $\lambda = 2$ remain in the originating cell.

This directly leads to more accurate Monte Carlo transport sweeps. We demonstrate this with the following experiment. Consider computing a transport sweep for the first group of the LRA-BWR problem in which the right hand side is a constant unit source. In all of the algorithms we consider, the transport sweep is required in order to compute the $\hat{D}$ term in order to build a consistent low-order problem. For this reason, we will compare the errors in $\hat{D}$ term for standard and CED tallies for two different particle counts. The results are displayed in Table 5.2.

This table clearly demonstrates the effectiveness of the CED tallies. The $\hat{D}$ terms and boundary conditions are the only quantities that are supplied by Monte Carlo transport sweep to the low-order problem. For this reason, measuring the error in $\hat{D}$ is the appropriate metric for choosing a tallying method.

Table 5.2: Error in $\hat{D}$ Computed Using MC Sweeps

	$\hat{D}_x$ Error		$\hat{D}_y$ Error	
Particles	CED	Standard	CED	Standard
10^{10}	.005009	.184261	.004974	.166200
10^{11}	.001598	.063798	.001461	.057308

Chapter 6

Hybrid Methods for Neutron Transport

In the beginning of Chapter 5 we discussed several reasons why Monte Carlo simulations may be preferable to deterministic solutions of the neutron transport equation. These stochastic methods can produce more accurate solutions, but potentially at a much higher cost. For this reason, we seek methods that will allow us to accelerate the convergence of Monte Carlo simulations. In Appendix C, we numerically investigate the truncation error in the low-order equation and the Monte Carlo error in the high-order transport sweep.

In the neutronics community, *variance reduction* techniques for Monte Carlo simulations have been studied thoroughly for as long as Monte Carlo algorithms have been used to simulate reactor physics [21]. Variance reduction refers to methods used to decrease the variance in a simulation without increasing the number of particles in the simulation. For example, in a transmission problem in which we wish to see what percentage of particle penetrate a very large, highly-absorbing medium, we may force particles to scatter and adjust particle weights accordingly. This allows a higher percentage of the particles in the simulation to contribute to the tally.

In this chapter, we'll attempt to build algorithms that allow us to compute solutions to the transport equation with the accuracy of Monte Carlo simulations, while reducing the total number of particles in the simulation and employing variance reduction techniques to lower the noise in the solution. However, we will not build these algorithms from scratch. Instead, we will utilize the algorithms developed in Chapter 2, Chapter 3 and Chapter 4 along with known variance reduction techniques to build hybrid algorithms for solving the neutron transport equation.

In the following sections we'll repeat the work done in Chapters 2, 3 and 4 using Monte Carlo transport sweeps. As we did earlier, we'll start with the 1-D fixed-source problem, as the

implementation is the most simple. We'll take what we learned from solving the fixed-source problem and accelerate the 1-D k-Eigenvalue problem. After this, we'll repeat this work in 2-D.

6.1 Accelerating a 1-D Fixed Source Monte Carlo Computation

In this section we'll consider hybrid methods for solving the 1-D fixed-source problem. The methods we'll consider are Picard-NDA(MC), AA-NDA(MC), and JFNK-NDA(MC)[1]. In general, we find that the noise in Monte Carlo transport sweeps can make using these acceleration methods difficult than when they were used with deterministic transport sweeps.

6.1.1 Noisy Function Evaluations

As we have previously discussed, Monte Carlo transport sweeps return noisy scalar fluxes and currents. In general, we measure the amount of noise in a Monte Carlo transport sweep by allowing the MC sweep to act on a deterministic solution to the transport equation, ϕ^*, and compute some measure of the difference $\phi^* - G_{MC}(\phi^*)$. In Figure 6.1, we see the results of a Monte Carlo transport sweep using a standard track-length tally and using the CED tally.

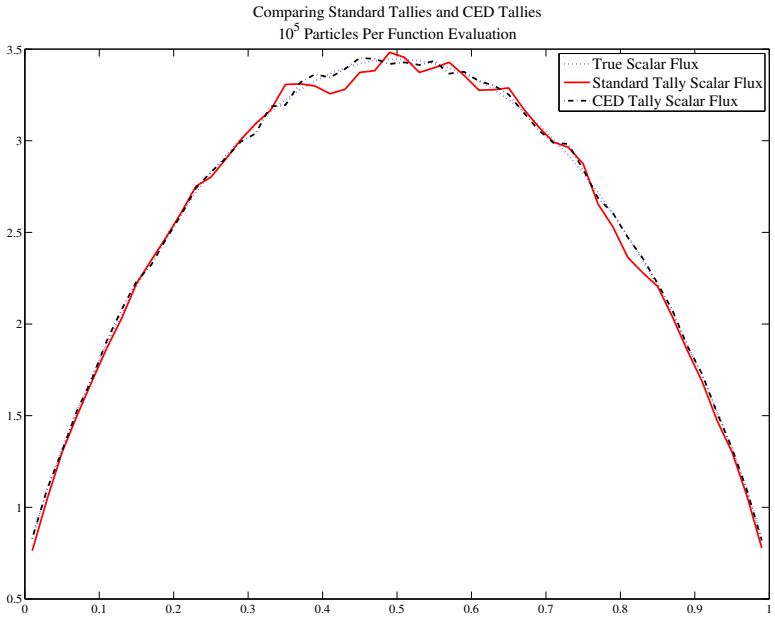

Figure 6.1: Comparing the scalar flux computed by a Monte Carlo transport sweep.

[1]The (MC) denotes that all transport sweeps involved in the algorithm are performed using a Monte Carlo simulation

As we can see in Figure 6.1, the CED tally performs slightly better than the standard track-length tally, however there is still a considerable amount of error, especially for $.25 \leq x \leq .75$. These oscillations in the flux may make it very difficult to compute an accurate consistency term, $\hat{D}$, for NDA. The noise in the Monte Carlo transport sweep is not limited to the scalar flux, however. In Figure 6.2, we see that the current suffers from a similar difficulty.

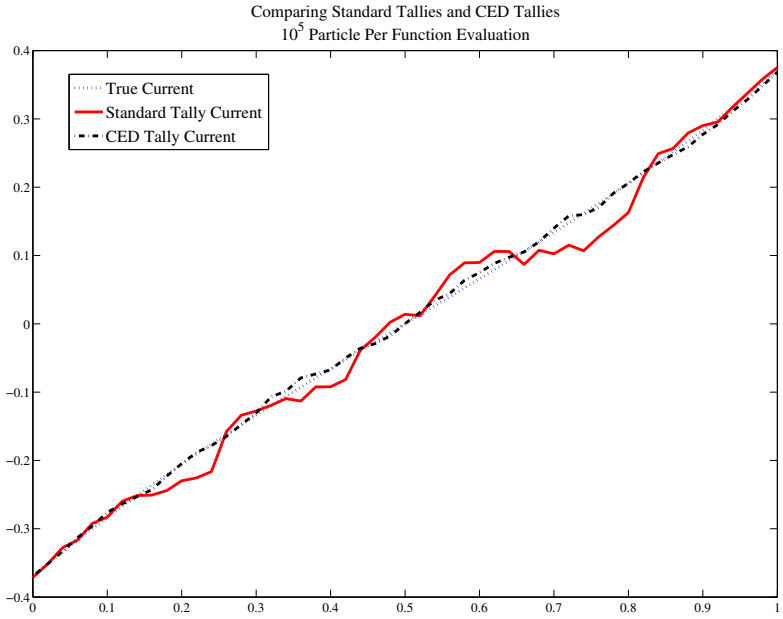

Figure 6.2: Comparing the current computed by a Monte Carlo transport sweep.

In Figure 6.2, we can see a much more drastic difference between standard tallies and the CED tally. In this case the CED tally accumulates much less error than standard cell-crossing tallies. It turns out that in some instances neither of these tallies are sufficiently accurate for computing $\hat{D}$. In Figure 6.3, we compare consistency terms computed using both the standard and CED tally.

It is apparent how destructive these noisy $\hat{D}$ functions can be when we consider the trying to compute the nonlinear residual defined by the JFNK-NDA problem. In Figure 6.4 we see the result of trying to compute $F(\phi^*)$, which should be nearly zero at the solution. Instead, it appears that we are nowhere close to finding a solution.

Unfortunately, increasing the number of particles per function evaluation does not necessarily solve this problem completely, as we can see in Table 6.1. Due to time constraints, increasing

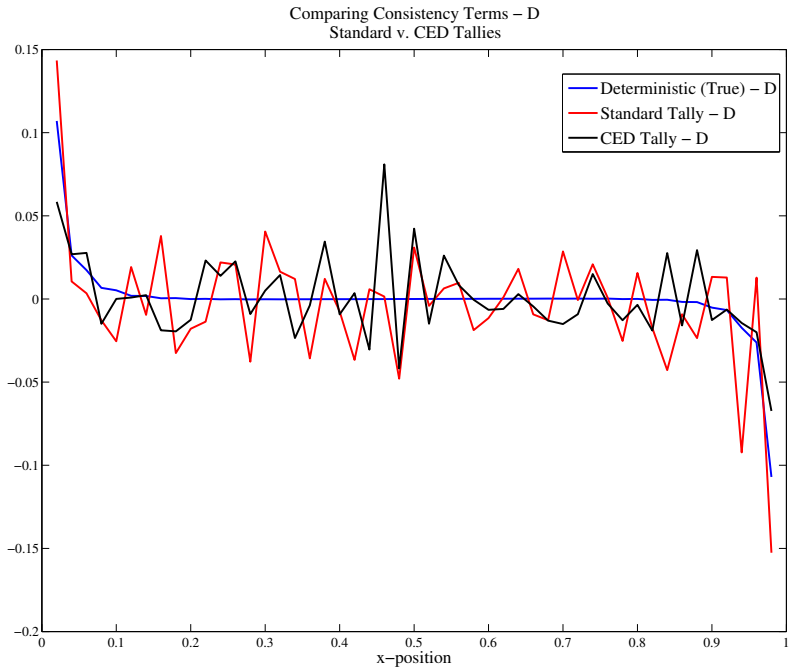

Figure 6.3: Comparing the consistency term computed by a Monte Carlo transport sweep.

the number of particles enough to resolve the flux and current to the desired level of accuracy is generally not feasible. In the next section, we'll consider two methods for filtering the noisy scalar flux and current.

Filtering

When using hybrid methods, we will make an attempt to remove some of the error from the results of a stochastic transport sweep before performing further computations and we'll refer to these error-removal processes as *filtering*. In general, the idea is to remove the low-amplitude,

Table 6.1: Comparing the Norm of the NDA Nonlinear Residual for Different Particle Counts and Tally Methods

Histories	Standard Tally	CED Tally
10^5	7.9358	7.2072
10^6	2.1414	1.8945
10^7	.8708	.8408
10^8	.4598	.4613

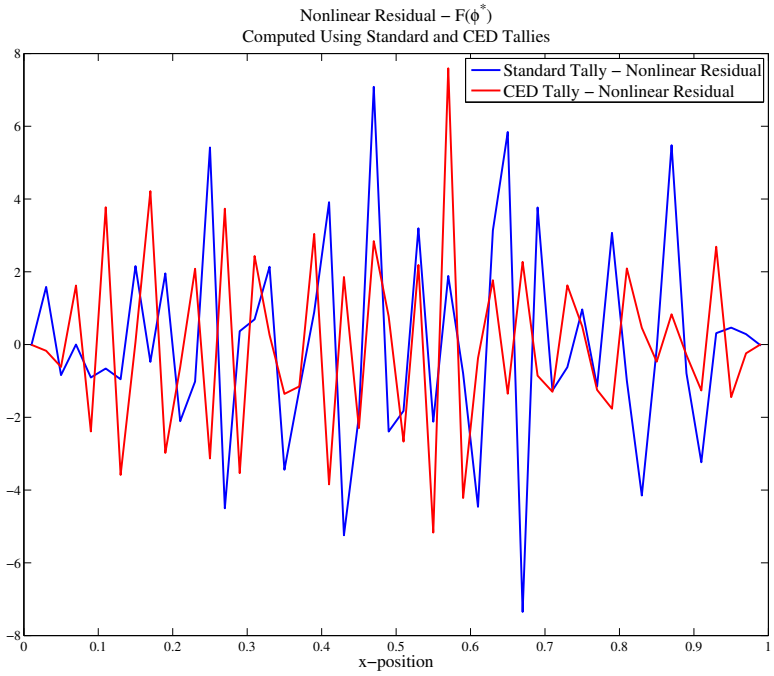

Figure 6.4: Comparing the JFNK-NDA nonlinear residual in which a Monte Carlo transport sweep is used.

high-frequency noise from the quantity of interest without changing any of the spatial structure that truly exists. This can be a difficult task in general, but in neutronics we often can determine where our solution may have special spatial structure based upon the material properties. We'll consider two similar filtering techniques in the remainder of this section.

For our first filtering method, which we will refer to as the *least squares spline filter* or LSS filter, let us divide our domain into a total of $R+1$ regions:

$$\{[0, x_1], [x_1, x_2], ..., [x_{R-1}, x_R], [x_R, X]\}$$

Within each of these regions, let us construct a polynomial of low degree that interpolates the function as closely as possible. We'll denote these degrees d_1, d_2, ... , d_R, d_{R+1} and we'll also demand that at each of the region boundaries x_i that the spline is continuous and has k_i continuous derivatives. By construction, we must choose $0 \leq k_i \leq \min(d_i, d_{i+1})$. We can perform a least squares fit of this low-degree polynomial spline to create a continuous, differentiable approximation to either the flux or current.

In Figure 6.5, we see a sample flux returned from a Monte Carlo function evaluation. We can smooth this function by using a least squares interpolation with regions separated by locations $x_1 = .1$, $x_2 = .3$, $x_3 = .7$ and $x_4 = .9$. Over each of the regions we use degree 3 polynomials and

demand that the function and its first derivative are continuous. We can see what a remarkable improvement this is over the noisy flux.

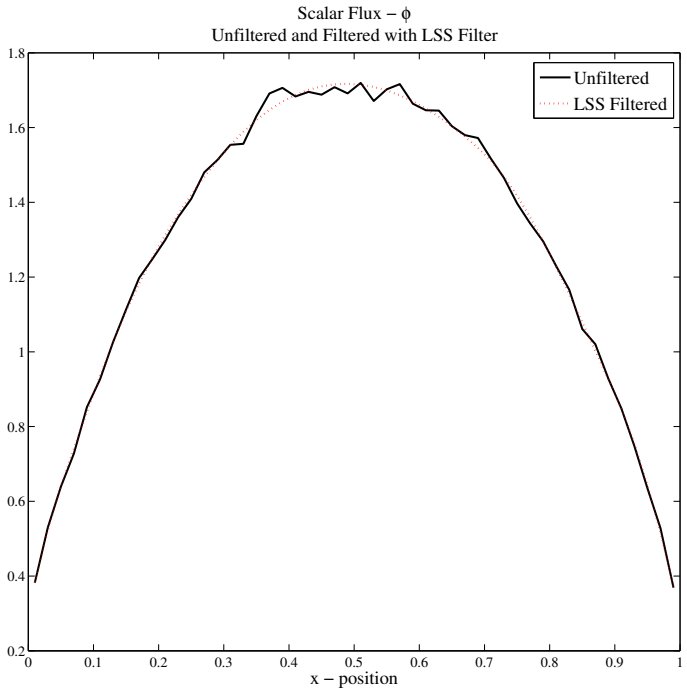

Figure 6.5: Noisy scalar flux before interpolation and after a least squares spline fit.

It is important to note that we must use low-degree polynomials for this interpolation. In fact, we would much prefer to use more regions each with equipped with a low-degree polynomial than a few regions with higher degree polynomials. When we begin to use high-degree polynomials, the polynomials tend to adapt to the shape of the noise and this is precisely what we are trying to avoid. By using all low-degree polynomials, our spline is forced to capture the overall shape of the curve and ignore the high-frequency noise.

In Figure 6.6, we consider the derivative of the two functions in Figure 6.5. We can see here how important this interpolation truly is. In some regions (for example $x \in [.4, .6]$), the true derivative is approximately 0, however the derivative of the noisy flux from Monte Carlo is nearly equal to 2. Errors this large would make it nearly impossible to solve $F(\phi) = 0$ using Newton's method (or any other nonlinear solution algorithm).

The second filtering method, which we will refer to as the *multiple-splining filter* or MS

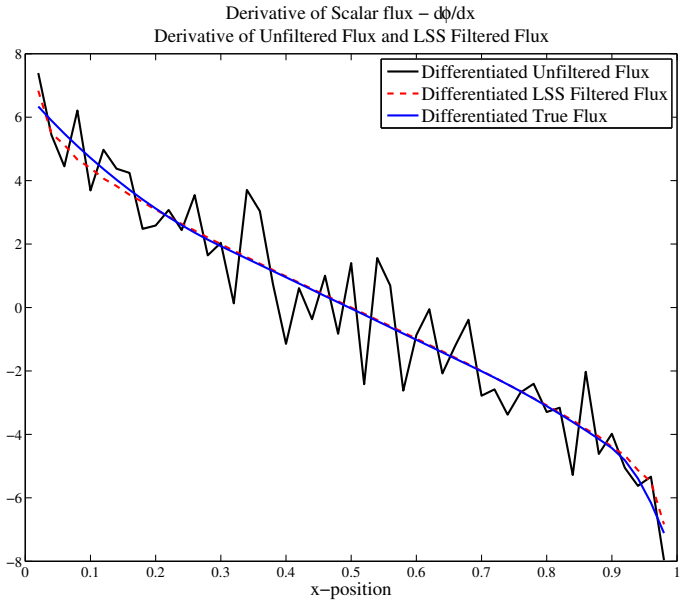

Figure 6.6: Comparing the finite difference derivative of the scalar flux before and after LSS filtering.

filter. For this filter, let us isolate a single point, (x_k, ϕ_k), from our unfiltered scalar flux. Now, we take some set of points around this point, $\vec{x} = [x_{k-m}\ x_{k-m+1}\ \dots\ x_k\ \dots\ x_{k+m-1}\ x_{k+m}]^T$ and $\vec{\phi} = [\phi_{k-m}\ \phi_{k-m+1}\ \dots\ \phi_k\ \dots\ \phi_{k+m-1}\ \phi_{k+m}]^T$. Through this set of $2m+1$ points we'll compute a least squares fit of a polynomial $p_k(x)$. Finally, we update $\phi_k^{NEW} = p_k(x_k)$. We repeat this for each of the points in our domain.

This has an advantage over the LSS filter; if we know a location for which there exist some rapid change is spatial structure, we can turn off the filter at that point or use points on only one side of the point for the filter. In Figure 6.7 we can see the effects of the MS filter. In Figure 6.8 we find the MS filter allows us to compute a more accurate derivative. The MS filter also has the advantage that it can be executed in parallel for each point in the domain.

Thinking forward to 2-D problems, it should be clear that the LSS filter would be very expensive and difficult to implement. With the LSS filter, we need to build a least squares problem which is the same size as the number of spatial cells. Furthermore, the constraints on region boundaries will be tricky to implement. On the other hand, the MS filter has no difficult constraints to include in the least squares problem. Furthermore, the size of the least squares optimization is far smaller. We have to solve one of these smaller least squares problems at each spatial point, but these can be trivially evaluated in parallel. All things considered, whenever a filter is called for in 2-D, we'll utilize only the MS filter.

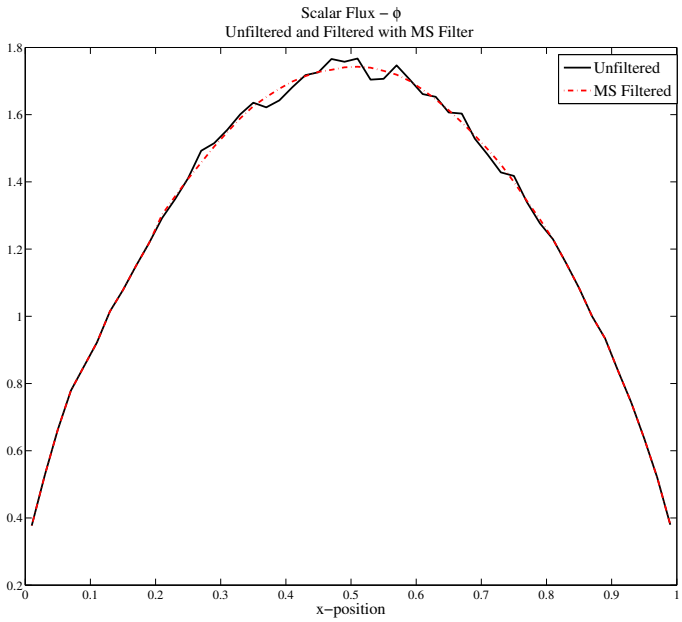

Figure 6.7: Noisy scalar flux before interpolation and after a multi-spline filter application.

6.1.2 Picard-NDA(MC)

In this section we explore Picard-NDA using Monte Carlo high-order function evaluations. We will see that convergence of this method is highly dependent on the number of particles per function evaluations. In Figure 6.9 we see the results of Picard-NDA(MC).

We see that without filtering and 10^6 particle per function evaluation, the residual stagnates around 2×10^{-2} or 3×10^{-2}, where the residual is computed by

$$R = \left\| \phi^{(n+1)} - \phi^{(n)} \right\|_{\infty}.$$

When using 10^9 particles per function evaluation we find that the nonlinear residual drops another order of magnitude before stagnation. At this point, the amount of error in the Monte Carlo transport sweep is on the same order of magnitude as the nonlinear residual. It is hopeless to try to converge the solution to a level below that of the error found in the transport sweep.

We can also compute the error for the scalar flux found using NDA(MC). We plot this error in Figures 6.10. This also shows that there is a strong connection between the error and the residual computed.

At this point, we can attempt to increase the number of particles per function evaluation, but this becomes impractical. On a compute cluster at Los Alamos National Laboratory we can

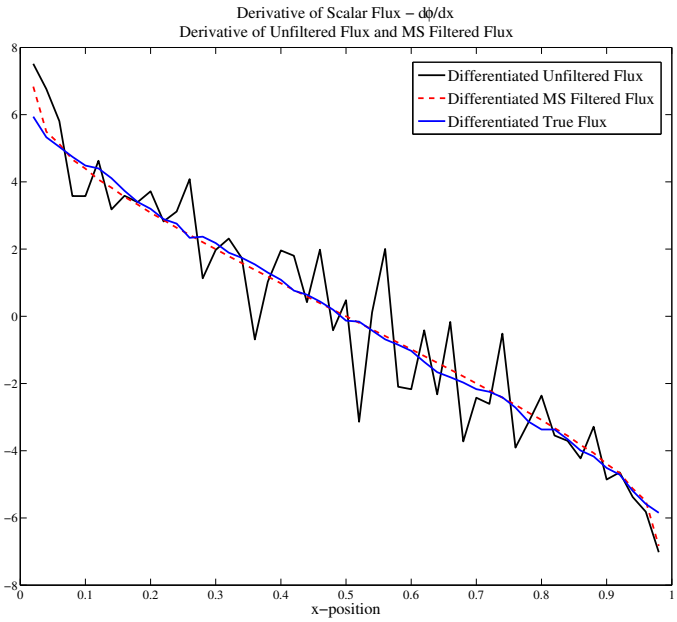

Figure 6.8: Comparing the finite difference derivative of the scalar flux before and after MS filtering.

simulate roughly 6.2×10^6 particles per second per core in MATLAB in serial. Using 8 cores, this amounts to 4.96×10^7 particles per second in parallel. Therefore, in order to execute a transport sweep using 10^9 particles it requires 20 seconds. Increasing the number of particles per transport sweep by a factor of 10 requires a factor of 10 increase in time. While 10^{10} particles per function evaluations (roughly 3 minutes) or 10^{11} particles per function evaluation (roughly 33 minutes) may be practical, using many more particles than this becomes a burden. Instead of increasing particle counts further, we turn to other methods in an attempt to decrease the noise in the stochastic transport sweep.

6.1.3 AA-NDA(MC)

In this section we'll explore the idea of applying Anderson Acceleration to NDA in which the transport sweeps are computed using Monte Carlo simulation. The basic idea here is that applying Anderson Acceleration will act as a sort of variance reduction in the sense that we take a linear combination of previous iterates in forming the new iterate. This has the potential to damp the noise found in function evaluations. Furthermore, we showed in Chapter 2 that Anderson Accelerated NDA performs much better than Picard NDA in some situations (see Table 2.2).

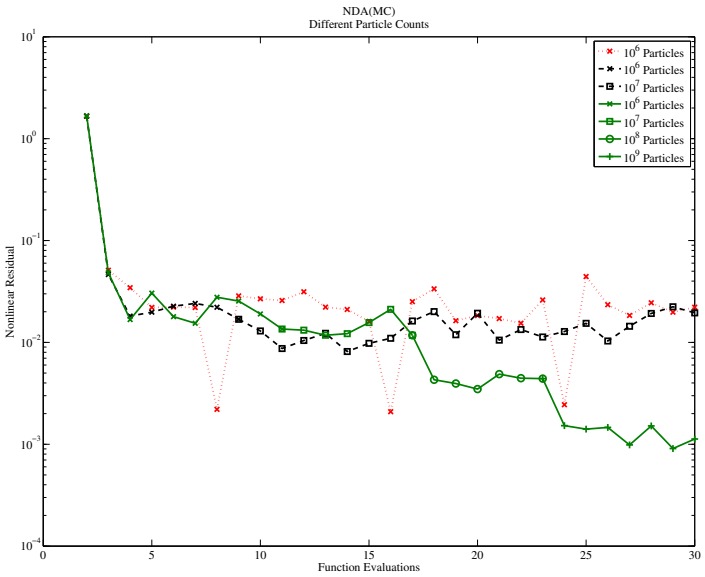

Figure 6.9: Comparing the convergence of NDA(MC) using varying numbers of particle per function evaluation.

As we can see in Figure 6.11, adding Anderson Acceleration does not allow us to lower the residual much below those computed for NDA(MC). In these figures, AA(1)-NDA corresponds to Picard-NDA, which is equivalent to Anderson-Acceleration when only the previous iterate is stored.

In Figure 6.11, we consider simulations in which we begin with relatively few particles and increase the number of particles by a factor of 10 every time the iteration stagnates. This allows us to use fewer particles at early stages in the iteration when less accuracy is needed. Unfortunately, including more vectors in the Anderson Acceleration history does not have a significant impact on the convergence rate. In the rest of this thesis, we abandon Anderson Acceleration for Newton-based methods.

6.1.4 JFNK-NDA(MC)

In this section we discuss an implementation of JFNK-NDA using Monte Carlo transport sweeps. With the previous algorithms, it was possible to see some convergence without filtering of any sort. However, with JFNK-NDA(MC), we are not so lucky. There are a couple of fundamental reasons why Monte Carlo transport sweeps pose a large problem for JFNK-NDA:

1. **Differentiating Noisy Functions:** In several places throughout the evaluation of F we

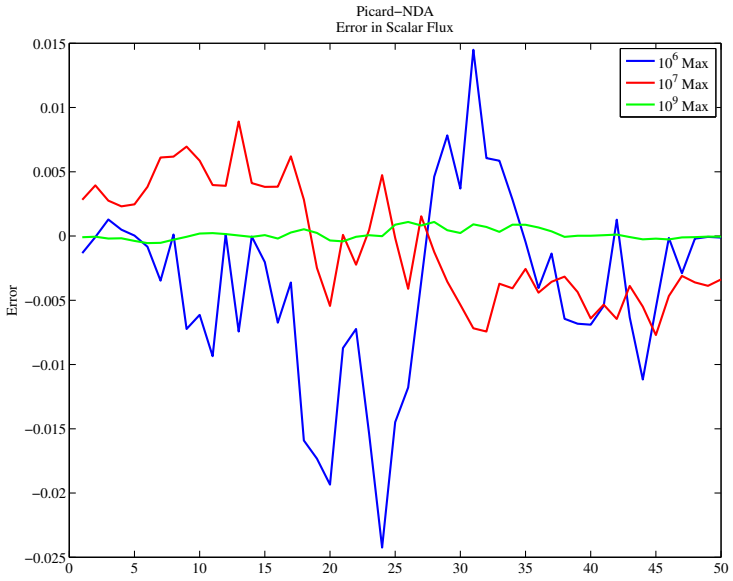

Figure 6.10: Comparing the error in scalar fluxes computed using differing particle counts in NDA(MC)

are required to differentiate a noisy function. The first instance where this comes into play is in the computation of $\hat{D}$. Within $\hat{D}$, we compute $\frac{d\phi}{dx}$ using a finite difference derivative. Later, when it comes time to actually evaluate F, we must compute the derivative of $\hat{D}$. This essentially means that we need to compute a second derivative of ϕ and a first derivative of J. Using finite differences to evaluate the derivative of a noisy function can extremely dangerous.

2. **Evaluating the Jacobian-Vector Product:** Within Newton's method, we must solve the linear equation $F'(\phi)v = -F(\phi)$ to compute the update v where $F'(\phi)$ is our Jacobian. In many applications, including this one, direct computation of the Jacobian is far too expensive to be a reasonable option. Instead, we can compute the action of the Jacobian matrix on a vector. That is, instead of determining the matrix $F'(\phi)$, we define a function that returns $F'(\phi)v$. This is often done using a forward difference derivative [15]:

$$F'(\phi)v \approx \frac{F(\phi + \epsilon v) - F(\phi)}{\epsilon} \tag{6.1}$$

For the reason mentioned above, using a finite difference derivative to approximate the Jacobian-vector product will not be a practical choice. Not only will we suffer the errors

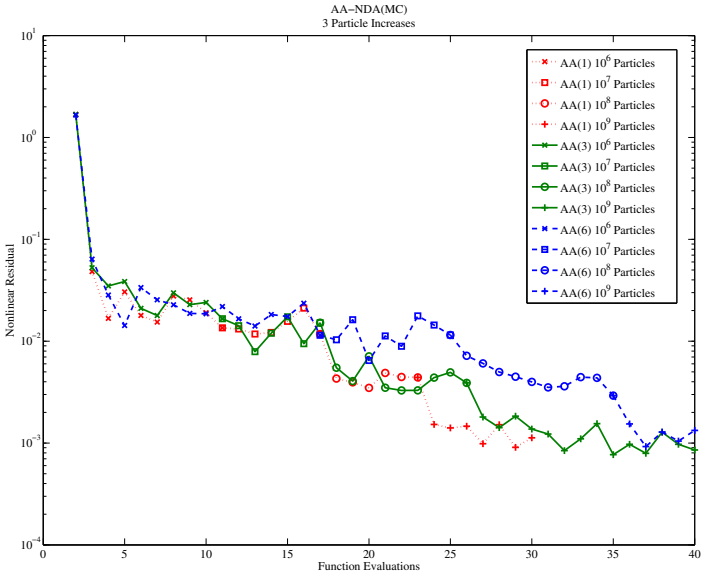

Figure 6.11: Comparing the residuals computed using AA-NDA(MC)

in the Jacobian-vector product evaluation, but we may have deeper issues when trying to use Krylov methods to solve the linear system.

Handling the noisy function evaluations is not necessarily easy, but it is straightforward. We have one of two options. First, we can increase the number of particles per function evaluation until the flux and current returned by the MC transport sweep have a level of error that allow us to compute $\hat{D}$ to our desired level of accuracy. As we've mentioned previously, this may be far too expensive. The second option is filtering, as was discussed in Section 6.1.1.

The second issue, noisy Jacobian-Vector products, has a very elegant solution. Instead of using the finite difference formula defined by Equation 6.1, we develop an analytic formula for the Jacobian-vector product or "jacvec" as we displayed in [32].

Analytic Jacobian-Vector Product

Within the Newton-Krylov iteration, we must solve Eq. 6.2 for the Newton step, v,

$$F'(\phi)v = -F(\phi) \tag{6.2}$$

where $F'(\phi)$ is the Jacobian.

Since F is Fréchet differentiable, we can utilize the following formula to compute the analytic Jacobian-vector product:

$$F'(\phi)v = \frac{d}{dh}F(\phi + hv)\bigg|_{h=0}. \tag{6.3}$$

In order to facilitate this calculation, we rewrite $F(\phi)$ using operator notation,

$$F(\phi) = \mathcal{L}\phi + \mathcal{N}\phi,$$

where $\mathcal{L}$ is the linear map,

$$\mathcal{L}\phi = \frac{d}{dx}\left[\frac{-1}{3\Sigma_t}\frac{d\phi}{dx}\right] + (\Sigma_t - \Sigma_S)\phi,$$

and $\mathcal{N}$ is the nonlinear map,

$$\mathcal{N}\phi = \frac{d}{dx}\left[\hat{D}(\phi^{HO}, J^{HO})\phi\right] - q.$$

Now, we can write the Jacobian-vector product in terms of its linear and nonlinear components

$$F'(\phi)v = \mathcal{L}'(\phi)v + \mathcal{N}'(\phi)v. \tag{6.4}$$

It is clear that $\mathcal{L}'(\phi)v$ is given by the simple expression

$$\mathcal{L}'(\phi)v = \mathcal{L}v. \tag{6.5}$$

The expression for $N'(\phi)v$ is somewhat more complicated. Using the Formula 6.3 and properties of differentiation (product rule), it is easy to show that

$$N'(\phi)v = \frac{d}{dx}\left[\hat{D}'(\phi)v\phi + \hat{D}(\phi)v\right] \tag{6.6}$$

where we have expressed $\hat{D}(\phi^{HO}, J^{HO})$ as $\hat{D}(\phi)$ to make the dependence of $\hat{D}$ on ϕ explicit.

Now, we can complete the formula by finding an expression for $\hat{D}'(\phi)v$. We accomplish this using the same technique we used to derive the expression for the nonlinear term. We find that

$$\hat{D}'(\phi)v = \frac{\phi^{HO}\left[J'^{HO}(\phi)v + \frac{1}{3\Sigma_t}\frac{d\phi'^{HO}(\phi)v}{dx}\right] - \left[J^{HO}(\phi) + \frac{1}{3\Sigma_t}\frac{d\phi^{HO}(\phi)}{dx}\right]\phi'^{HO}(\phi)v}{(\phi^{HO}(\phi))^2} \tag{6.7}$$

where

$$\phi'^{HO}(\phi)v = \int_{-1}^{1} \xi(x,\mu)d\mu \tag{6.8}$$

$$J'^{HO}(\phi)v = \int_{-1}^{1} \mu\xi(x,\mu)d\mu. \tag{6.9}$$

Here, $\xi(x,\mu)$ is the solution to the external-source-free transport equation where the scalar flux is taken to be $v(x)$,

$$\mu\frac{\partial\xi(x,\mu)}{\partial x} + \Sigma_t\xi(x,\mu) = \frac{1}{2}\Sigma_s v(x).$$

By combining Equations 6.4-6.9 we have an analytic formula for the Jacobian-vector product.

It should be clear that much like the finite-difference approximation to the Jacobian-vector product, we can compute the analytic Jacobian-vector product with only a single additional transport sweep. This means that, at no added cost to us, we have a formula for a more accurate representation of the Jacobian-vector product.

In practice, we find that when the transport sweep is computed deterministically and the mesh is relatively fine, the analytic and finite-difference Jacobian-vector product agree to several decimal places. However, when the Monte Carlo transport sweep is used, these two quantities can differ by orders of magnitude when the number of particles per function evaluation is small.

Algorithmic Adjustments

In general, we'd like to solve the transport equation using as few particles as possible. For this reason, we start using a minimal number of particles per function evaluation. At some point, the the error in the function evaluations becomes large enough that convergence stalls and we encounter an Armijo line-search failure. We consider two options for managing the increase of particles as the iteration progresses.

Option 1

Our original method of increasing the number of particles per function evaluation demanded that we start with some relatively small number of particles per transport sweep, N_{MC}. Then, each time we encounter an Armijo failure, we

- increase N_{MC} by a factor of 100 and
- restart the Newton-GMRES iteration from the previous iterate.

We must establish some notation before we formally describe the algorithm. We approximate $F(u)$ with a Monte Carlo simulation to obtain $F(u, N_{MC})$ and the Jacobian-vector product

$F'(u)w$ with a Monte Carlo simulation with output $J(u, w, N_{MC})$ where N_{MC} denotes the number of particles per MC transport sweep. The evaluations of F we need for the Newton step and testing termination tolerances for the nonlinear iteration, or within the line search in the Newton-GMRES code must be replaced with calls to the Monte Carlo simulation.

Newton-GMRES-MC$_1$

Evaluate $R_{MC} = F(u, N_{MC})$; $\tau \leftarrow \tau_r \|R_{MC}\| + \tau_a$.
while $\|R_{MC}\| > \tau$ **do**
  Use GMRES with a limit of I_{max} iterations to find d such that $\|J(u, d, N_{MC}) + R_{MC}\| \leq \eta \|R_{MC}\|$
  if the GMRES iteration fails **then**
    $N_{MC} \leftarrow 100 * N_{MC}$
    Evaluate $R_{MC} = F(u, N_{MC})$
  else
    $\lambda = 1$
    Evaluate $R_{Trial} = F(u + \lambda d, N_{MC})$
    while $\|R_{Trial}\| > (1 - \alpha\lambda)\|R_{MC}\|$ and $\lambda \geq \lambda_{min}$ **do**
      $\lambda \leftarrow \lambda/2$
      Evaluate $R_{Trial} = F(u + \lambda d, N_{MC})$
    end while
    if $\lambda \geq \lambda_{min}$ **then**
      $u \leftarrow u + \lambda d$
      $R_{MC} = R_{Trial}$
    else
      $N_{MC} \leftarrow 100 * N_{MC}$
      Evaluate $R_{MC} = F(u, N_{MC})$
    end if
  end if
end while

This is a simple modification of the Newton-GMRES algorithm and seems to work well [32]. However, when employing this method, we often waste a lot of particle histories evaluating the function in the line search or when the Newton iteration begins to stall. Furthermore, it is difficult to prove any convergence rate results when using this strategy for increasing the number of particles. For these reasons, we consider a second, more efficient option.

Option 2

In [30], we demonstrate that we can obtain r-linear convergence by increasing the number of particles per transport sweep at each iteration. It is important to note, that as long as we choose a suitable initial number of particles per Jacobian evaluation, this value never needs to increase throughout the iteration. Only the number of particles per function evaluation increases from iteration to iteration. This particle management scheme is described in Algorithm 6.1.4.

Newton-GMRES-MC$_2(u, N_{MC}, N_{MC}^J, \eta, \tau_r, \tau_a)$
  Evaluate $R_{MC} = F(u, N_{MC})$; $\tau \leftarrow \tau_r \|R_{MC}\| + \tau_a$.
  while $\|R_{MC}\| > \tau$ **do**
    Compute $J(u, N_{MC}^J)$
    Find s which satisfies $\|J(u, N_{MC}^J)s + \tilde{F}(u, N_{MC})\| \leq \eta \|\tilde{F}(u, N_{MC})\|$.
    $u \leftarrow u + s$
    Evaluate $R_{MC} = F(u, N_{MC})$;
    $N_{MC} \leftarrow N_{inc} N_{MC}$
  end while

This algorithm allows us to "track" r-linear convergence in which $r = \sqrt{N_{inc}}$ [30]. Compared to **Option 1**, this is a much more efficient use of particle histories. In Section 6.1.4, we'll see the outcome of using each of these particle increase methods.

Results

We present results here for JFNK-NDA(MC). We'll consider two test problems that demonstrate the convergence properties. In both test problems, both the flux and current are filtered using a least squares interpolant filter in both the function and Jacobian evaluation. Furthermore, during the function evaluation, $\hat{D}$ is filtered using the same least squares interpolant filter.

In each of the following simulations, we precondition the linear solver using the same preconditioner as described in [18]. We choose to use the inverse of the operator, M, defined by

$$M = \frac{\partial}{\partial x}\left[-\frac{1}{3\Sigma_t}\frac{\partial}{\partial x} + \hat{D}^k\right] + (\Sigma_t - \Sigma_s),$$

where $\hat{D}^k$ was computed during the most recent evaluation of F.

In order to test this algorithm and compare the two particle management strategies, we'll use the following parameters:

Parameter	Value
Σ_t	10
Σ_s	9.9
τ	1
q	.5
Spatial Cells	50

Option 1

In Figure 6.12, we see that using 10^6 particles per function evaluations (using the standard tally) does not provide enough accuracy to converge to the desired tolerance. However, using 10^6 particles per function evaluation does allow us to get a rough shape for the scalar flux. We use this estimate of the flux as the starting point for a new run of JFNK-NDA(MC) using 10^8 particles per function evaluation. This provides yet a better estimate for a third pass of the algorithm.

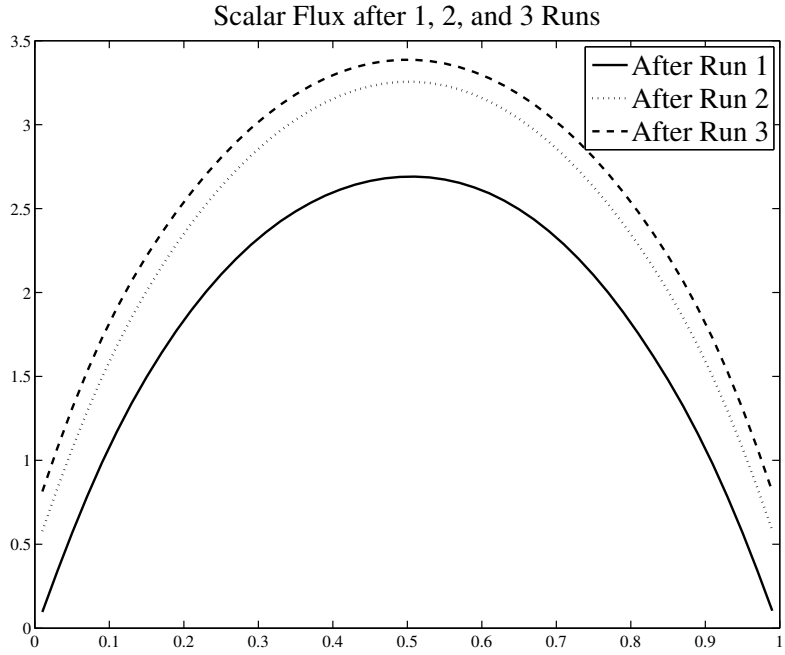

Figure 6.12: Fluxes after 1, 2 and 3 runs of JFNK-NDA(MC) with increasing particle counts using standard tallies.

In Figure 6.13, we see that using 10^6 particles per function evaluations provides an order of magnitude decrease in the nonlinear residual from the initial iterate for both standard and CED tallies. Increasing the number of particles by a factor of 100 allows for another order of magnitude decrease. The final run, using 10^{10} particles per function evaluation drops the norm of the nonlinear residual below 5×10^{-3}.

Comparing the standard and CED tallies, we see that the CED tallies are more effective at lower particle counts. By the end of Run 2, the CED tallies had decreased the nonlinear residual by an extra factor of two over the standard tallies and had done so using few function evaluations. At the level of 10^{10} particles per function evaluation, the two tallying methods become indistinguishable.

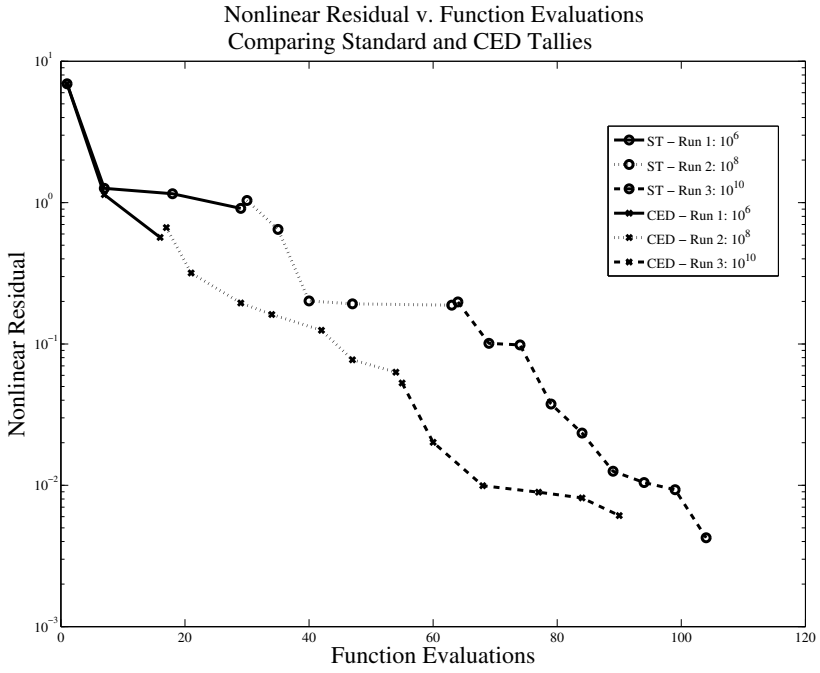

Figure 6.13: Nonlinear residuals after 1, 2 and 3 runs of JFNK-NDA(MC) with increasing particle counts using both Standard and CED Tallies.

Option 2

With **Option 2**, there are several parameters which we can change. We fix $N_{MC}^J = 10^6$ for the entire simulation. We vary the quantities K_L (max number of Krylovs per Newton step), η (inexact Newton forcing term), and K_R (number of times that GMRES is initiated per Newton

step). When $K_R = 1$, GMRES with at most K_L iterations is used to compute the Newton step. When $K_R = 2$, GMRES computes an approximation to the Newton step, and then GMRES is called a second time using the approximation to the Newton step as the initial iterate. This is similar to GMRES with restarts, however GMRES is called a second time regardless of the linear residual at the end of the first call to GMRES. This helps to alleviate issues caused by noisy Jacobian evaluations in the early stages of the GMRES iteration [30]. We allow the forcing term to take on values $\eta = .1$, .01, and .001. K_L can take on values 5 and 10.

We display the numerical results when using the second method of managing particles and vary the quantities K_L, K_R and η in Figures 6.14 through Figures 6.25. In each of the following figures we employ the same structure. On the y-axis we plot the nonlinear residual, on the x-axis we plot the total number of accumulated particle histories (realizations), and we plot the results of ten simulations along with a dashed line demonstrating a rate of residual decrease that tracks $\frac{1}{\sqrt{N_{MC}}}$.

In Figures 6.14 and 6.15 we see the effect that a second pass of GMRES can have on the overall iteration. In Figure 6.14 the nonlinear residual history is a considerably smoother function of the number of total realizations. We see a similar trend in Figures 6.16 and 6.17. In each of these figures we hold η constant at .001.

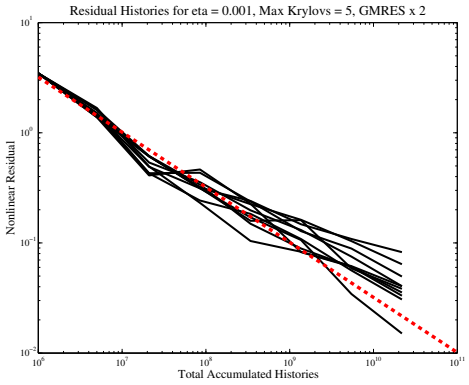

Figure 6.14: $K_L = 5$, $\eta = .001$, $K_R = 2$

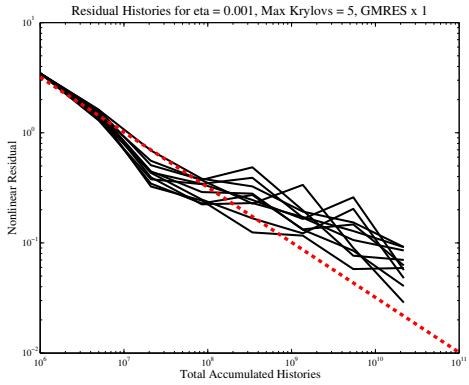

Figure 6.15: $K_L = 5$, $\eta = .001$, $K_R = 1$

A similar trend persists for $\eta = .01$. By the end of the iteration, we see that the nonlinear residual is generally slightly lower when GMRES is restarted ($K_R = 2$). We can see this by comparing Figures 6.18 and 6.19 and Figures 6.20 and 6.21.

Finally, let us consider $\eta = .1$. When $K_L = 10$, we see very little difference between restarting or not. However, when $K_L = 5$, the difference is noticeable. In this case restarting is better in the sense that the simulations are both more consistent over the ten trials and

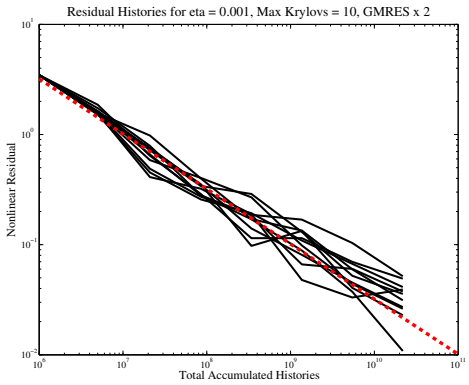

Figure 6.16: $K_L = 10$, $\eta = .001$, $K_R = 2$

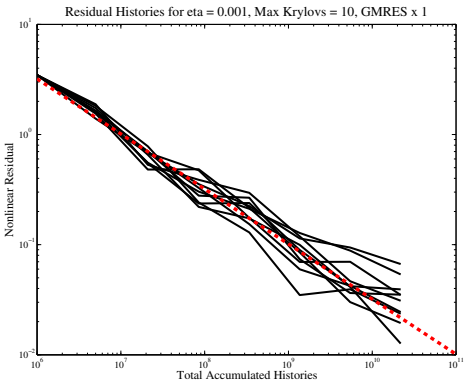

Figure 6.17: $K_L = 10$, $\eta = .001$, $K_R = 1$

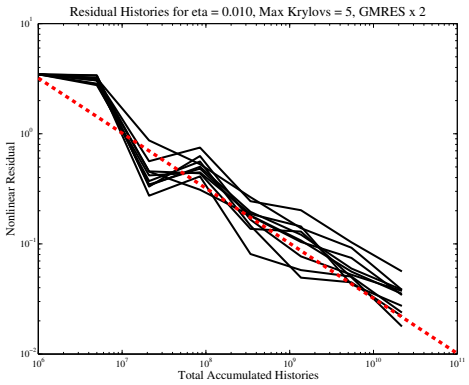

Figure 6.18: $K_L = 5$, $\eta = .01$, $K_R = 2$

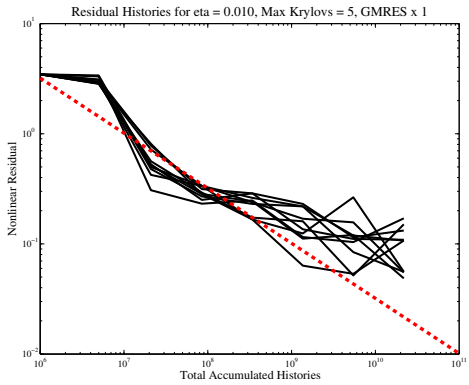

Figure 6.19: $K_L = 5$, $\eta = .01$, $K_R = 1$

the residuals decrease more rapidly. This should be no surprise from the previous test results. These results are demonstrated in Figures 6.22 – 6.25.

In Figures 6.14 - 6.25 we see that, in general, restarting GMRES leads to more predictable residual histories. When GMRES is restarted to compute the Newton step, the residual histories for all the simulations are very similar. When GMRES is not restarted, this is not always the case. For example, the residual histories in Figure 6.23 have a large variance.

All of the figures in this section demonstrate that we can track r-linear convergence by properly increasing the number of particles per function evaluation. In order to reach a nonlinear residual of 2×10^{-2}, this particle management strategy with an appropriate choice of the parameters K_L, K_R, and η requires roughly 2×10^{10} histories. Compare this to the first strategy for increasing the number of particles. With CED tallies, it still requires at least

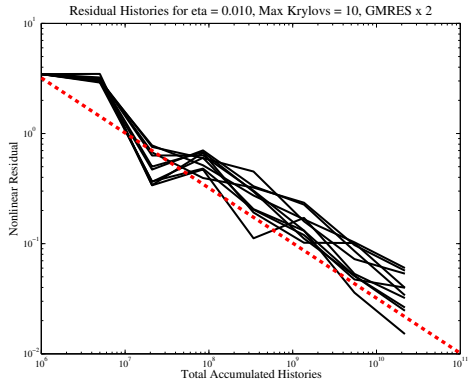

Figure 6.20: $K_L = 10$, $\eta = .01$, $K_R = 2$

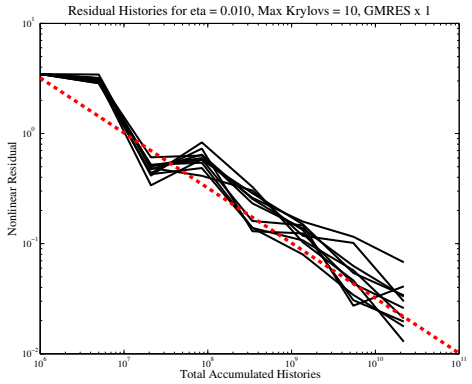

Figure 6.21: $K_L = 10$, $\eta = .01$, $K_R = 1$

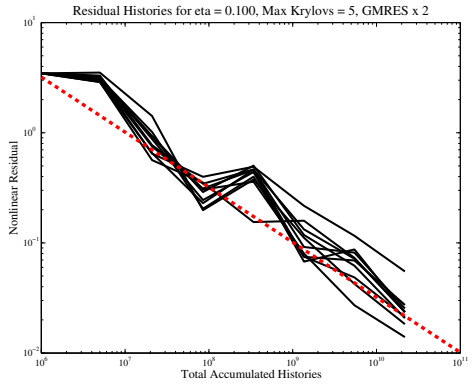

Figure 6.22: $K_L = 5$, $\eta = .1$, $K_R = 2$

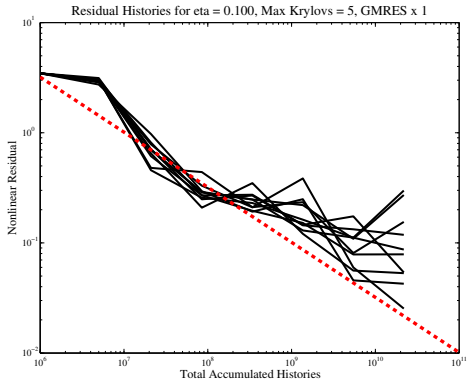

Figure 6.23: $K_L = 5$, $\eta = .1$, $K_R = 1$

three times as many particles to decrease the residual to this point. Using standard tallies, it required roughly 15 times as many particles to low the residual to 2×10^{-2}. Clearly, increasing the number of particles per function evaluation is a far superior strategy than waiting for the iteration to stall. For this reason, we will adopt **Option 2** for the remainder of this thesis for both fixed-source problems and k-eigenvalue problems.

6.2 Accelerating a 1-D k-Eigenvalue Monte Carlo Computation

In Chapter 4 we explored methods for solving the neutron transport k-eigenvalue problem. We found that we could accelerate standard power iterations using NDA-NCA, in which much of the computational work was moved to the low-order diffusion problem. Much like we did in

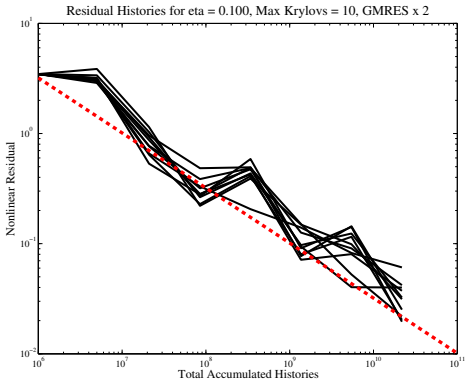

Figure 6.24: $K_L = 10$, $\eta = .1$, $K_R = 2$

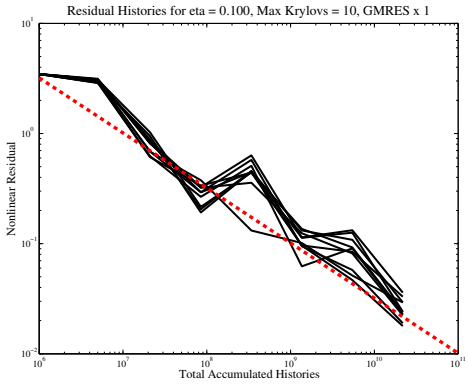

Figure 6.25: $K_L = 10$, $\eta = .1$, $K_R = 1$

Section 6.1, we'd now like to consider hybrid methods for solving the k-eigenvalue problem using Monte Carlo transport sweeps [33].

It should be clear from the computational results in Chapter 4 and Section 6.1 that standard power iteration using Monte Carlo transport sweeps would be infeasible. The increased computational effort of using Monte Carlo transport sweeps demands that we reduce the total number of transport sweeps which we need to compute. NDA-NCA provides an excellent way to do this. Not only does it reduce the number of transport sweeps considerably, it moves much of the work to low-order diffusion solves which are both inexpensive and deterministic. This means that the only source of stochastic error comes from the transport sweep calculated in the outer iteration. Contrast this to JFNK-NDA(MC) in which both the residual evaluations and the Krylov solve contributed to the noise when trying to solve the fixed-source problem.

Furthermore, it should be clear at this point that NDA-NCA should be used in place of NDA-PI. While NDA-PI is a drastic improvement over unaccelerated power iteration, it cannot compete with NDA-NCA in terms of computational effort required to update the eigenvalue and eigenvector within each outer iteration. When the Newton step in NDA-NCA is computed using the Sherman-Morrison formula to invert the Jacobian [31], the cost of updating the eigenvalue/eigenvector pair is only two sparse matrix inversions (low-order diffusion solves). We ask the reader to recall the results from Table 4.2 at the end of Chapter 4.

In Section 6.1.4 we demonstrated that using a Newton-Krylov method to solve $F(\phi) = 0$ when F is evaluated stochastically can be quite difficult. We should be careful to note that this is not the case with NDA-NCA. While there is an element of stochasticity in the computation of $\hat{D}$, this value is not computed within the evaluation of F. Within JFNK-NDA(MC) each evaluation of F at a vector ϕ yielded a slightly different result based around the expected

value of $F(\phi)$. In NDA-NCA, $F(\phi)$ is computed exactly (in the absence of discretization error, machine roundoff, etc.) each time. Therefore, we need not take quite as much care when solving the $F(\phi) = 0$. The only source of noise in this algorithm is introduced in the transport sweep at the beginning of each outer iteration.

6.2.1 Results

We will consider two 1-D, 1-Group test problems to demonstrate the effectiveness of this algorithm. Both Test 1 and Test 2 below consist of a single fissile region surrounded by two 5 mean-free-path reflectors [20]. The length of the medium is given by τ, and the length of the fissile region is given by $\tau - 10$.

Test 1

Consider the parameters for the first test:

$$\begin{aligned} \Sigma_t = 1 \quad \Sigma_s = .856 \quad \nu\Sigma_f = .144 \quad \tau = 35 \quad \text{(fissile region)} \\ \Sigma_t = 1 \quad \Sigma_s = .856 \quad \nu\Sigma_f = 0 \quad\ \ \tau = 35 \quad \text{(reflector region)} \end{aligned}$$

For this problem, we found the true eigenvalue to be $k_{eff} = .9720469427$, computed using a high accuracy S_n computation. We'll run the simulation 10 times in order to gauge performance. For each simulation we'll discretize using 7000 spatial cells. We begin using 10^6 particles per function evaluation and increase the number of particles per function evaluation by a factor of 2 each iteration. After 15 outer iterations, we have used a total of roughly 3.28×10^{10} particle flights. After 10 independent simulations we collect the following results for k_{eff}:

- Mean = **.9720469**297
- Standard deviation = .0000028874
- Max eigenvalue = **.9720**577602 (.001866% error)
- Min eigenvalue = **.97204**20155 (.001850% error)

Test 2

Consider the parameters for the second test:

$$\begin{aligned} \Sigma_t = 1 \quad \Sigma_s = .856 \quad \nu\Sigma_f = .144 \quad \tau = 60 \quad \text{(fissile region)} \\ \Sigma_t = 1 \quad \Sigma_s = .856 \quad \nu\Sigma_f = 0 \quad\ \ \tau = 60 \quad \text{(reflector region)} \end{aligned}$$

For this problem, the true eigenvalue is $k_{eff} = .9919893631$. We'll run the simulation 10 times in order to gauge performance. For each simulation we'll discretize using 12000 spatial cells. We begin using 10^6 particles per function evaluation and increase the number of particles per function evaluation by a factor of 2 each iteration. After 15 outer iterations, we have used a total of roughly 3.28×10^{10} particle flights. After 10 independent simulations we collect the following results for k_{eff}:

- Mean = **.991989**2935
- Standard deviation = .0000011455
- Max eigenvalue = **.9919**907012 (.000142% error)
- Min eigenvalue = **.99198**75238 (.000178% error)

A comparison against an analog Monte Carlo computation is given in Appendix D.

6.2.2 Conclusions

We have demonstrated that using Monte Carlo transport sweeps within the NDA-NCA algorithm is feasible. All of these results have been obtained without filtering of any sort. We believe that filtering will not be necessary for the k-eigenvalue problem. The computation of k involves an integral over the domain which acts as a filter by using errors in one part of the domain to cancel out errors in other parts of the domain.

This gives us confidence that the NDA-NCA(MC) algorithm works and can be tested in 2-D. It is expected that more particles per transport sweep will be required in 2-D, so we will briefly discuss our parallel implementation of the MC transport sweep in `C++` in Section 6.3.

6.3 2-D Monte Carlo Transport Sweep

Up until this point, all of the Monte Carlo computations have taken place in MATLAB. While MATLAB has been sufficient for 1-D, we understand that moving up in dimension will require significantly more particles per transport sweep. For this reason, we have written a 2-D Monte Carlo transport sweep in `C++` which has been parallelized using both MPI and OpenMP. The low-order solver still takes place in MATLAB. The communication between MATLAB and `C++` takes place via file input/output. This allows us to continue to use the code base that has been developed for the deterministic 2-D computations, without significant headache.

At each call to the `C++` transport sweep, the work is spread out between the total number of nodes available, N_T, using MPI. This means that if the sweep calls for N_{MC} particle histories, each node will be responsible for computing N_{MC}/N_T particle histories. Each node reads in

Table 6.2: Weak Scaling Efficiency for Standard and CED Tallies 10^9 Particles Histories Per Node

	Standard Tally		CED Tally	
Nodes	Time	Efficiency	Time	Efficiency
1	19.34	——	85.17	——
2	19.42	99.7%	85.22	99.9%
4	19.63	98.6%	85.50	99.6%
8	20.05	96.50%	85.67	99.4%
12	20.18	95.9%	85.72	99.4%
20	20.53	94.25%	86.08	98.9%

the source term, the total cross-sections for the entire domain, and a few domain parameters. Right now, the domain is replicated on each node. If the domain were to become so large that storing the entire domain on a single node became troublesome, domain decomposition could be used without much trouble. In this case, each node would be responsible for simulating histories in a small fraction of the domain.

Once the work has been spread out to each node via MPI, we use OpenMP on each node to spread out the work over each core. Again, each core receives the total cross-sections for the entire domain. If N_C cores are available per node, then the amount of work per core amounts to $N_{MC}/(N_T \times N_C)$ particle histories. After each core has finished executing its batch of particles, we average the scalar flux and current in parallel using OpenMP within each node. The final scalar flux and current are computed via an MPI_Reduce on the master node. At this point, the scalar flux and current are written to a text file to be interpreted by MATLAB.

We have performed a scaling study on the Monte Carlo transport sweep on a computing cluster at Los Alamos National Laboratory. This compute cluster is comprised of nodes which each contain four AMD Opteron 6168 processors running 12 CPU cores at 1.90 GHz. We use 20 nodes ($N_T = 20$) and each node has 48 cores ($N_C = 48$). We measure the time it takes to compute a single group 1 transport sweep for this study. The results of this study can be seen in Figure 6.26 and Table 6.2.

As expected, we have very efficient both strong and weak scaling. As the amount of work per node increases, these scaling numbers become even more impressive. This highly parallel Monte Carlo Transport Sweep will allow us to solve the fixed-source and k-eigenvalue problems in 2-D in a reasonable amount of time.

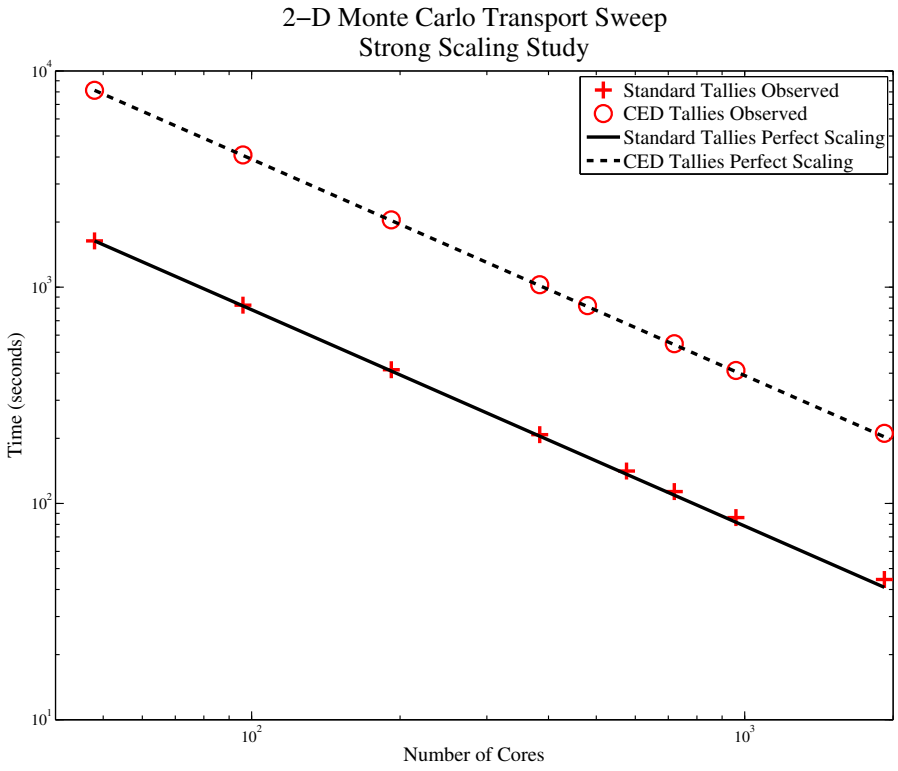

Figure 6.26: Both Standard and CED tallies achieve very efficient strong scaling across 40 nodes. 10^{11} particles per transport sweep were used.

6.4 Accelerating a 2-D Fixed Source Monte Carlo Computation

In 2-D, we will only consider using JFNK-NDA(MC) to solve the fixed-source problem. We'll use what we learned in Section 6.1 to guide our development of the 2-D code. We know that the first key to effectively solving the transport equation with JFNK-NDA(MC) is the use of the analytic Jacobian-vector product. We will derive this formula in detail in the following section. The second key to effectively solving the transport equation with JFNK-NDA(MC) is the proper management of particles. We will use only **Option 2** from Section 6.1 in which the number of particles per transport sweep are held constant for Jacobian-vector product evaluations and increased each iteration for the residual function evaluation. The combination of these ideas should lead to a smooth transition from deterministic 2-D JFNK-NDA to a functioning hybrid implementation.

6.4.1 2-D Analytic Jacobian-Vector Product

Just as in 1-D, we'll derive the analytic Jacobian-vector product for the NDA equations. In this case, we'll derive the equations in full generality, ignoring dimension, and allowing for a multi-group formulation.

We'll remind the reader of the multi-group formulation of the fixed-source neutron transport problem

$$\hat{\Omega} \cdot \nabla \psi_g + \Sigma_{t,g} \psi_g = \frac{1}{4\pi} \left[\sum_{g'=1}^{G} \Sigma_s^{g' \to g} \phi_{g'} + Q_g \right],$$

in which g ranges from 1 to G. It will be important to recall the definition of the scalar flux and current, as well,

$$\begin{aligned} \phi_g &= \int_{4\pi} \psi_g d\hat{\Omega} \\ \vec{J}_g &= \int_{4\pi} \hat{\Omega} \psi_g d\hat{\Omega}. \end{aligned}$$

We use Newton's method to solve the following nonlinear system of equations,

$$F_g(\Phi) = \nabla \cdot \left[-\frac{1}{3\Sigma_t} \nabla \phi_g + \hat{D}_g \phi_g \right] + \left(\Sigma_{t,g} - \Sigma_s^{g \to g} \right) \phi_g - \sum_{g' \neq g} \Sigma_s^{g' \to g} \phi_{g'} - Q_g$$

in which $\hat{D}_g$ is a function of Φ and $\Phi = [\phi_1 \ \phi_2 \ \dots \ \phi_G]^T$.

Within Newton's method, we must solve the following linear system of equations for the Newton step, s,

$$F'(\Phi) s = -F(\Phi),$$

in which F' denotes the Jacobian of the function F. Traditionally, $F'(\Phi)v$ is computed using a finite-difference derivative,

$$F'(\Phi) v \approx \frac{F(\Phi + \epsilon v) - F(\Phi)}{\epsilon},$$

for some small choice of ϵ. For settings in which the function evaluation $F(\Phi)$ may be low accuracy (e.g. evaluated using Monte Carlo transport sweeps), the finite-difference derivative is generally very inaccurate. We seek to derive an analytic formula for the Jacobian-vector product.

To do so, let us first re-write the function, F, in terms of its linear and nonlinear components

$$F_g(\Phi) = \mathcal{L}_g\Phi + \mathcal{N}_g(\Phi) - Q_g,$$

in which

$$\begin{aligned}
\mathcal{L}_g\Phi &= \nabla\cdot\left[-\frac{1}{3\Sigma_t}\nabla\phi_g\right] + \left(\Sigma_{t,g} - \Sigma_s^{g\to g}\right)\phi_g - \sum_{g'\neq g}\Sigma_s^{g'\to g}\phi_{g'}, \\
\mathcal{N}_g(\Phi) &= \nabla\cdot\hat{D}_g(\Phi)\phi_g.
\end{aligned}$$

Now, we can write the Jacobian of F in terms of its linear and nonlinear components,

$$F'(\Phi)v = \mathcal{L}v + \mathcal{N}'(\Phi)v.$$

The only remaining task is to compute the action of $\mathcal{N}'(\Phi)$ on a vector, v.

We'll use the following formula for the Jacobian vector product,

$$\mathcal{N}'(\Phi)v = \frac{d}{d\epsilon}\mathcal{N}(\Phi+\epsilon v)\Big|_{\epsilon=0}.$$

This yields

$$\begin{aligned}
\mathcal{N}'(\Phi)v &= \frac{d}{d\epsilon}\left[\nabla\cdot\hat{D}(\Phi+\epsilon v)(\Phi+\epsilon v)\right]\Big|_{\epsilon=0} \\
&= \nabla\cdot\frac{d}{d\epsilon}\left[\hat{D}(\Phi+\epsilon v)(\Phi+\epsilon v)\right]\Big|_{\epsilon=0} \\
&= \nabla\cdot\left[\left(\hat{D}'(\Phi)v\right)\Phi + \hat{D}(\Phi)v\right].
\end{aligned}$$

We can evaluate $\hat{D}'(\Phi)v$ in a similar manner. It is important to note that $\hat{D}$ is a function of Φ through the high order flux and current. $\vec{J}^{HO}$ and Φ^{HO} are functions of Φ through the transport sweep. Now, we have,

$$\begin{aligned}
\hat{D}'(\Phi)v &= \frac{d}{d\epsilon}\left[\frac{\vec{J}^{HO}(\Phi+\epsilon v) + \frac{1}{3\Sigma_t}\nabla\Phi^{HO}(\Phi+\epsilon v)}{\Phi^{HO}(\Phi+\epsilon v)}\right]\Big|_{\epsilon=0} \\
&= \frac{\frac{d}{d\epsilon}\left[J^{HO}(\Phi+\epsilon v) + \frac{1}{3\Sigma_t}\Phi^{HO}(\Phi+\epsilon v)\right]\Big|_{\epsilon=0}\Phi^{HO}(\Phi)}{\Phi^{HO}(\Phi)^2} \\
&\qquad - \frac{\frac{d}{d\epsilon}\Phi^{HO}(\Phi+\epsilon v)\Big|_{\epsilon=0}\left[J^{HO}(\Phi) + \frac{1}{3\Sigma_t}\nabla\Phi^{HO}(\Phi)\right]}{\Phi^{HO}(\Phi)^2} \\
&= \frac{\left[J^{HO}(\Phi)'v + \frac{1}{3\Sigma_t}\nabla\Phi^{HO}(\Phi)'v\right]\Phi^{HO}}{(\Phi^{HO})^2} - \frac{\Phi^{HO}(\Phi)'v\hat{D}(\Phi)}{\Phi^{HO}}
\end{aligned}$$

We will represent the i^{th} angular moment of Ψ^{HO} as $M^{i,HO}$. This way, we can deal with $\vec{J}^{HO}(\Phi)'v$ and $\Phi^{HO}(\Phi)'v$ simultaneously. Now, we can write

$$\begin{aligned} M^{i,HO}(\Phi)'v &= \frac{d}{d\epsilon}\left[M^{i,HO}(\Phi+\epsilon v)\right]\Big|_{\epsilon=0} \\ &= \frac{d}{d\epsilon}\int_{4\pi}\hat{\Omega}^i\Psi^{HO}(\Phi+\epsilon v)d\hat{\Omega}\Big|_{\epsilon=0} \\ &= \int_{4\pi}\hat{\Omega}^i\Psi^{HO}(\Phi)'v d\hat{\Omega}. \end{aligned}$$

Lastly, we know that Ψ^{HO} is related to Φ via the transport equation, which we will write as

$$\mathcal{A}\Psi^{HO} = \mathcal{S}\Phi + Q,$$

so

$$\Psi^{HO} = \mathcal{A}^{-1}\mathcal{S}\Phi + \mathcal{A}^{-1}Q.$$

Now, we have

$$\begin{aligned} \Psi^{HO}(\Phi)'v &= \frac{d}{d\epsilon}\left[\mathcal{A}^{-1}\mathcal{S}(\Phi+\epsilon v)+\mathcal{A}^{-1}Q\right]\Big|_{\epsilon=0} \\ &= \mathcal{A}^{-1}\mathcal{S}v. \end{aligned}$$

That is, $\Psi^{HO}(\Phi)'v$ can be computed by executing a single transport sweep in which v is used for the scattering source and the fixed source is taken to be zero.

We evaluate $F'(\Phi)v$ using the following steps:

1. Input Φ^{HO}, J^{HO}, and $\hat{D}$, which have already been computed.
2. Input Krylov vector, v.
3. Execute a single transport sweep to recover $\bar{\Psi} \equiv \Psi^{HO}(\Phi)'v = \mathcal{A}^{-1}\mathcal{S}v$.
4. Compute $\bar{\Phi} \equiv \Phi^{HO}(\Phi)'v$, $\bar{J} \equiv \vec{J}^{HO}(\Phi)'v$ using

$$\begin{aligned} \bar{\Phi} &= \int_{4\pi}\bar{\Psi}d\hat{\Omega} \\ \bar{J} &= \int_{4\pi}\hat{\Omega}\bar{\Psi}d\hat{\Omega} \end{aligned}$$

5. Evaluate $\bar{D} \equiv \hat{D}'(\Phi)v$

$$\bar{D} = \frac{\left[\bar{J} + \frac{1}{3\Sigma_t}\nabla\bar{\Phi}\right]\Phi^{HO}}{(\Phi^{HO})^2} - \frac{\bar{\Phi}\hat{D}}{\Phi^{HO}}$$

6. Evaluate $\mathcal{N}'(\Phi)v$

$$\mathcal{N}'(\Phi)v = \nabla \cdot \left[\bar{D}\Phi + \hat{D}v\right]$$

7. Evaluate $F'(\Phi)v$

$$F'(\Phi)v = \mathcal{L}v + \mathcal{N}'(\Phi)v.$$

With this analytic Jacobian-vector product in hand, we are ready to implement JFNK-NDA(MC) in 2-D. The next section describes results for a simple 2-D multi-material test problem.

6.4.2 2-D JFNK-NDA(MC) Results

We'll test JFNK-NDA(MC) on a 3 material 2-D, 2-group test problem. The material layout is portrayed in Figure 6.27. The corresponding cross-section data is displayed in Table 6.3. The properties of the domain are described in Table 6.4.

Table 6.3: Fixed-Source Problem Material Properties

Material	$\Sigma_{t,1}$	$\Sigma_{t,2}$	$\Sigma_s^{1\to1}$	$\Sigma_s^{1\to2}$	$\Sigma_s^{2\to2}$
1	0.2656	1.5798	0.2021	0.0253	1.4795
2	0.2623	1.7525	0.2280	0.0277	1.6821
3	0.2652	2.0938	0.2170	0.0475	2.0747

Table 6.4: Domain Properties for Fixed-Source Problem

τ_x	τ_y	N_x	N_y	χ_1	Boundaries
50	50	50	50	1	Reflective Corner

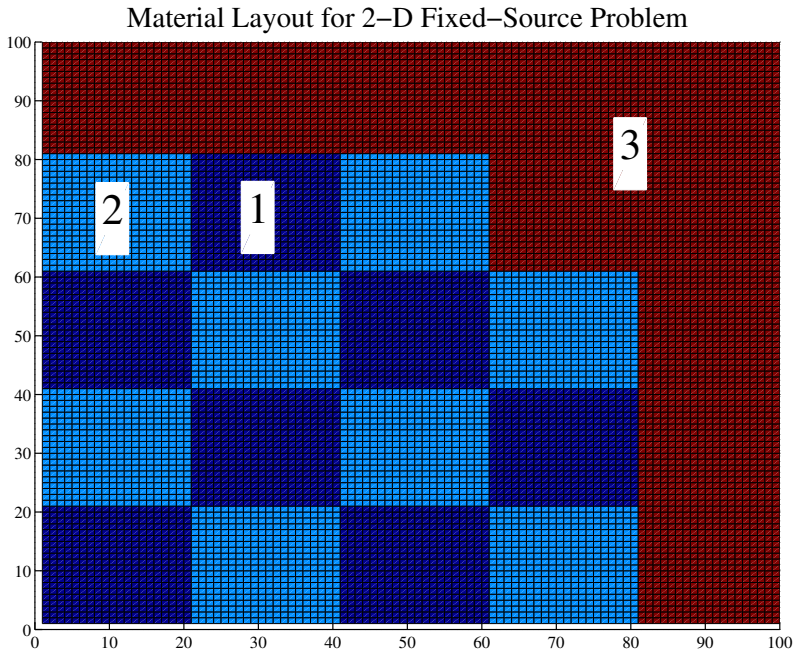

Figure 6.27: Material layout for 2-D Fixed-Source Problem.

In Figure 6.28 we plot the scalar flux for groups 1 and 2. In Figure 6.29 we plot the nonlinear residual as a function of iteration, which corresponds directly to the number of particles per function evaluation. As we can see, we are able to track r-linear convergence. For these simulations we use GMRES twice per iteration to compute the Newton step. We use $\eta = .01$ and $K_L = 10$ (number of Krylovs per GMRES solve). We see that our observations from 1-D hold again in 2-D and have allowed for a straightforward implementation.

6.5 Accelerating a 2-D k-Eigenvalue Computation

In 2-D, we'll utilize NDA-NCA(D) to compute the k-eigenvalue, just as in Section 6.2. However, before moving on, let us address the issue of convergence in hybrid k-eigenvalue computations.

With deterministic methods, we can define convergence in several obvious ways. For example, we can terminate when the relative difference between two successive values of the eigenvalue is less than some tolerance. We could also check the normed relative difference of the eigenvector. Furthermore, we could place convergence criteria on the norm of the low-order residual, F, in the NDA-NCA algorithm. Any of these checks can be justified as convergence

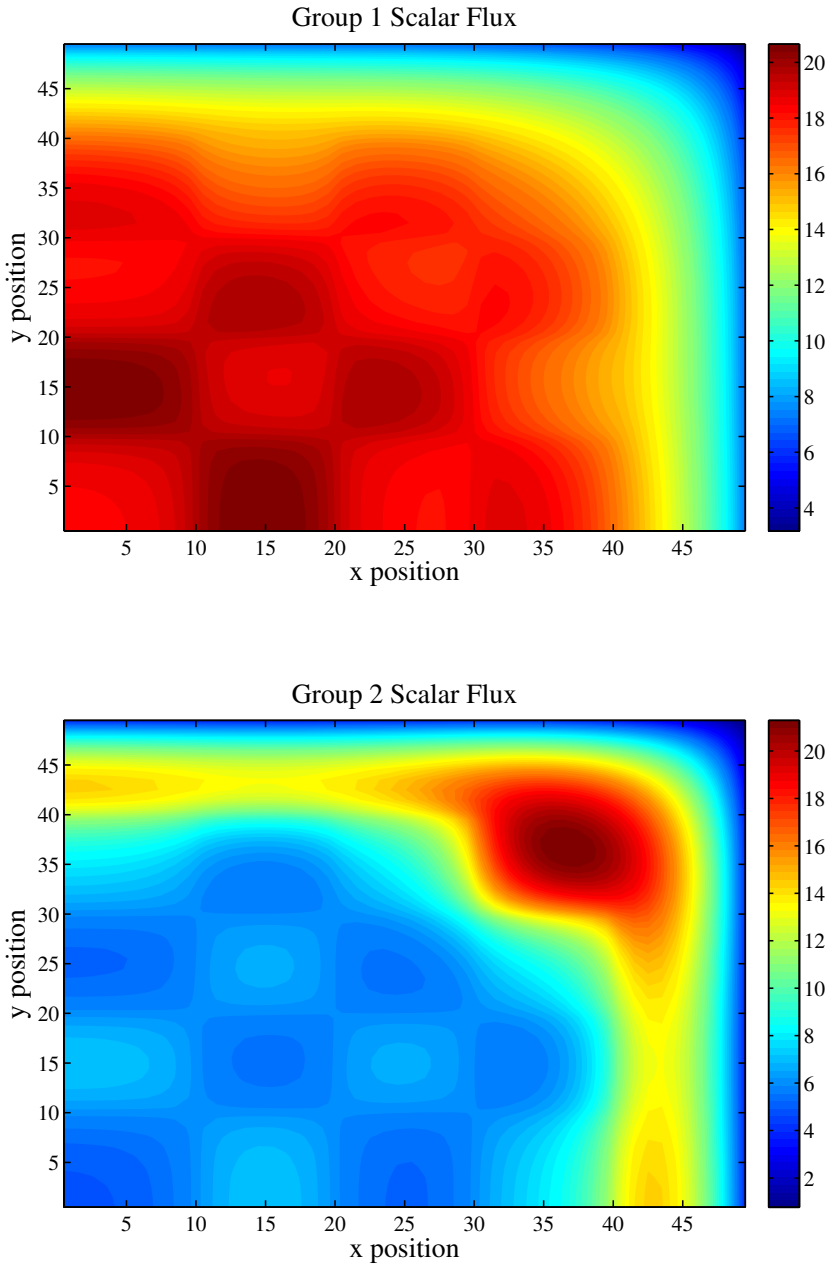

Figure 6.28: Hybrid solution to the 2-D Fixed-Source Problem.

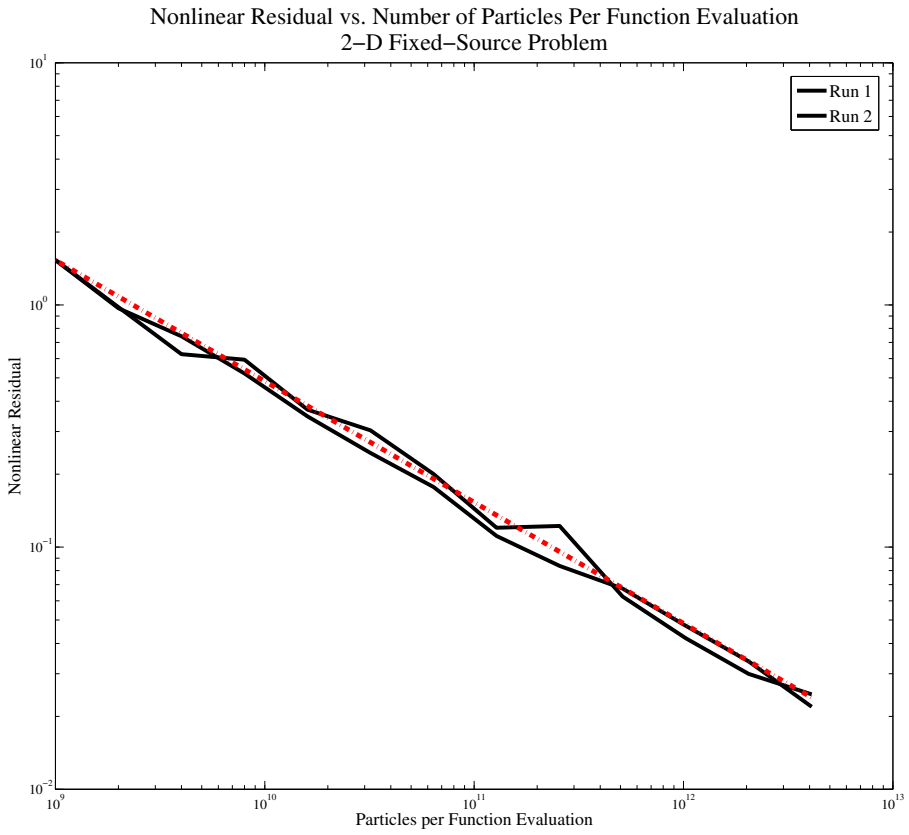

Figure 6.29: JFNK-NDA(MC) Convergence History for 2-D Fixed-Source Problem

criteria.

With purely stochastic methods, one can test the variance in the eigenvalue computation. Once the variance is below some specified value, we can consider the eigenvalue converged. With stochastic methods, due to the random nature of the computation, it can be dangerous to consider the iteration converged when two successive values of the eigenvalue are very close. Furthermore, due to the stochastic noise, it may be very expensive to iterate until the scalar flux is no longer changing.

With hybrid methods, the question as to what to use as a convergence criterion becomes somewhat clouded. The variance is a more difficult quantity to compute as we are no longer averaging "active cycles." Furthermore, there is a randomness associated with the eigenvalue and even the low-order residual, so placing a test on these quantities alone may be ill-advised. We will turn to a test of convergence that is becoming increasingly popular in the neutronics

community, a check of the *Shannon Entropy* [35, 25, 3].

6.5.1 Shannon Entropy

The Shannon Entropy is, in essence, a measure of the randomness in the fission source and is given by

$$H(S_f) = -\sum_{s=1}^{N_s} p_s \log_2(p_s), \tag{6.10}$$

in which H is the Shannon Entropy, S_f is the fission source, N_s is the number of fission sites which are labeled from 1 to N_s, and p_s is the percentage of the total fission reaction that occurs at fission site s [35, 3]. In order to compute these values, we look at the number of spatial cells (independent of group) in which the fission source exists. In each spatial cell we compute the fission source and compute the total global fission source. With these quantities in hand, we can compute the Shannon Entropy.

It is expected that this value converges and is a good measure of the convergence of the eigenvalue and eigenvector, as these values are intimately tied to the fission source. In Figures 6.30 and 6.31, we can see that for a deterministic calculation, the Shannon Entropy converges for both the TMR and the LRA-BWR problems, respectively.

It is also interesting to plot the change in the Shannon Entropy from one iteration to the next on the same axes as the error in the k-eigenvalue. Again, using the results of a deterministic calculation, these quantities can be seen for the two problems in Figures 6.32 and 6.33.

The fact that the change in Shannon Entropy for these problems is a good representation for the error in the eigenvalue is a convincing argument that it should be used as a convergence metric. In these 2-D computations k-eigenvalue computations, we will keep a history of the Shannon Entropy, the eigenvalue, the normed difference of successive iterates of the eigenvector and the norm of the low-order residual, F. In this way, we hope to show that the convergence of each of these values can be captured by considering the Shannon Entropy.

6.5.2 Numerical Results

In this section we will consider the TMR problem, the LRA-BWR problem and Zion reactor problem. For each of the problems, we will start with 10^9 particles per group transport sweep. At each iteration we will increase the number of particles by a factor of 2 until the eigenvalue has converged or we reach a total of 10^{13} particles. Starting at the fifth iteration, we will begin to average the scalar flux and current with previous iterates which will be used in the

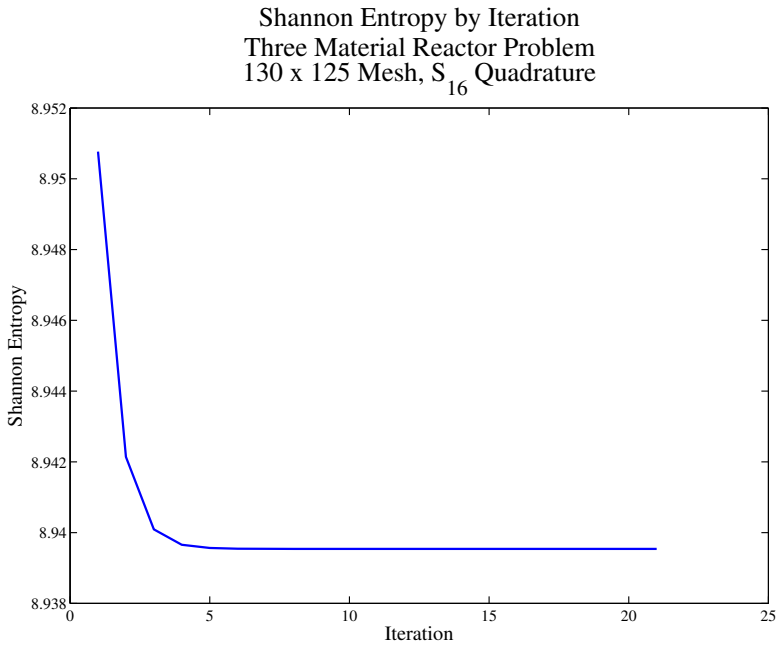

Figure 6.30: The Shannon Entropy converges for the TMR problem.

computation of $\hat{D}$. We will use the weighted average

$$\Phi^{(i+1)} = \frac{1}{3}\Phi^{(i)} + \frac{2}{3}\tilde{\Phi}^{(i+1)} \tag{6.11}$$

$$\vec{J}^{(i+1)} = \frac{1}{3}\vec{J}^{(i)} + \frac{2}{3}\tilde{J}^{(i+1)} \tag{6.12}$$

in which $\tilde{\Phi}$ and $\tilde{J}$ are the high-order quantities computed directly from the transport sweep at the most current iteration. We argue that this weighting scheme is justified because twice as many particles were used in the most recent transport sweep than the previous sweep. The correction term is not averaged directly, however it is computed using the averaged scalar fluxes and currents. By weighted averaging we utilize the history of the iteration instead of throwing it out completely, while putting the most emphasis on the most current, accurate simulations.

For each of these simulations, we will monitor four convergence metrics. These convergence metrics include:

- **Change in Shannon Entropy**: This quantity measures how much the Shannon entropy changes from one iteration to the next.
- **Change in k-eigenvalue**: This quantity measure how much the eigenvalue is changing

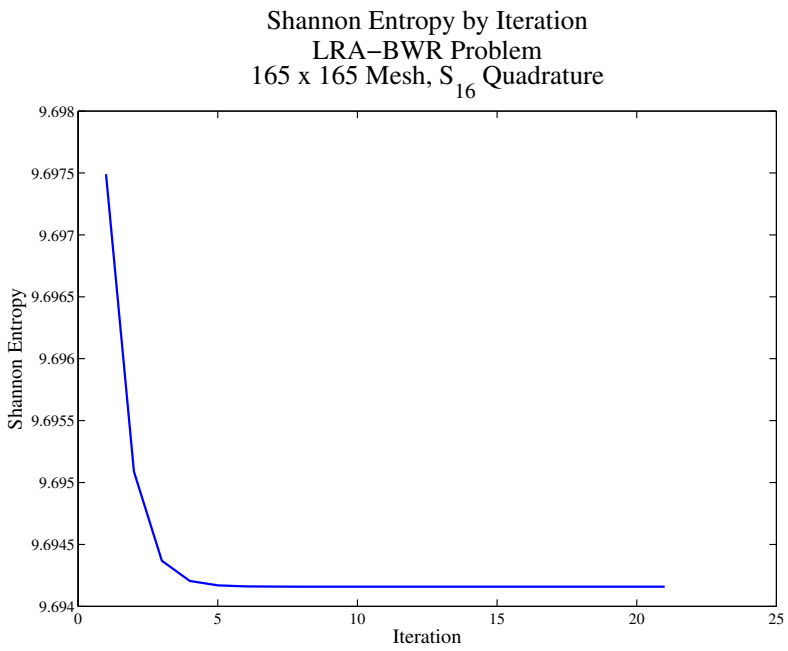

Figure 6.31: The Shannon Entropy converges for the LRA-BWR problem.

from iteration.

- **Change in eigenvector**: This quantity measures the relative normed difference of the eigenvalue between successive iterates.
- **Norm of nonlinear residual**: This quantity measure the nonlinear residual of the low-order problem at each iteration.

LRA-BWR

We have run two independent NDA-NCA(MC) simulations and compiled results. After 14 outer iterations and maximum of 8.192 $\times 10^{12}$ particles per function evaluation and using a 165 $\times$ 165 mesh, we have locked onto the first five digits of the eigenvalue

$$\begin{aligned} k_{eff,1} &= \mathbf{1.00021}9056020964 \\ k_{eff,2} &= \mathbf{1.00021}8814100412 \\ k_{eff,3} &= \mathbf{1.00021}8975804200 \end{aligned}$$

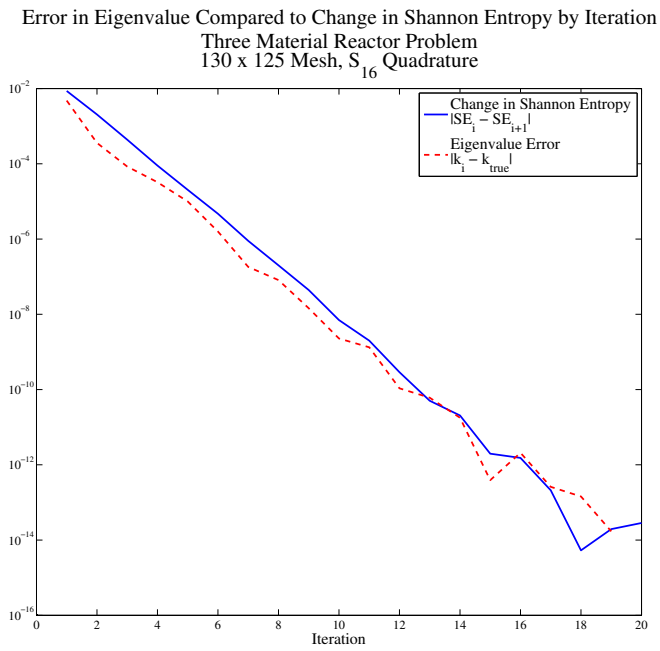

Figure 6.32: The change in Shannon Entropy from iteration to iteration goes to zero at the same rate as the error in the eigenvalue for the TMR problem.

in all three simulations. In Figure 6.34, we plot the eigenvector computed using the hybrid method alongside the difference between the eigenvector computed using hybrid methods and deterministic methods. By any of the four convergence metrics, we can see that the eigenvalue and eigenvector are converging.

In Figures 6.35 and 6.36, we look at a combination of the four convergence metrics for each simulation.

Three-Material Reactor

We have run three independent NDA-NCA(MC) simulations and compiled results. After 14 outer iterations and maximum of 8.192 $\times 10^{12}$ particles per function evaluation and using a 125 $\times$ 130 mesh, we have locked onto the first seven digits of the eigenvalue

$$\begin{aligned} k_{eff,1} &= \mathbf{1.143718}36029695 \\ k_{eff,2} &= \mathbf{1.143718}27718820 \end{aligned}$$

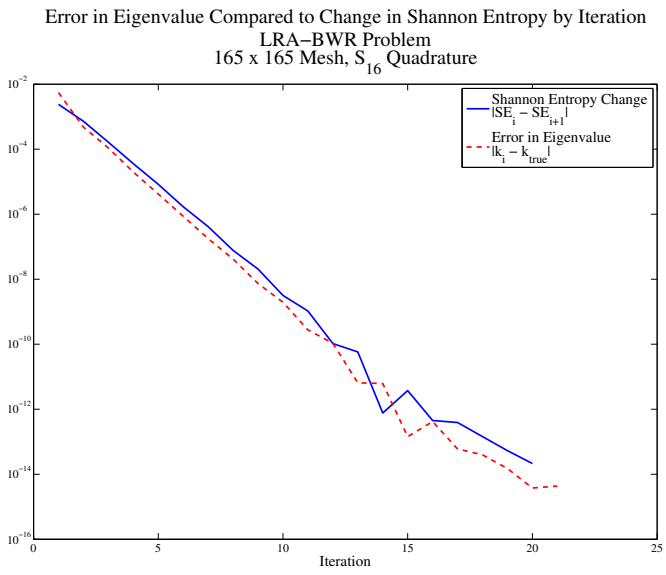

Figure 6.33: The change in Shannon Entropy from iteration to iteration goes to zero at the same rate as the error in the eigenvalue for the LRA-BWR problem.

in both simulations. In Figure 6.37, we plot the eigenvector computed using the hybrid method alongside the difference between the eigenvector computed using hybrid methods and deterministic methods.

In Figures 6.44 and 6.39, we look at a combination of the four convergence metrics for each simulation. It appears that the nonlinear residual is stalling, however the eigenvector still appears to be converging and both the eigenvalue and the Shannon entropy are converging. Furthermore, both the eigenvalue and eigenvector compare well with deterministic solutions.

We can gain some insight into the situation by considering the spatial dependence of the nonlinear residual. In Figures 6.40 and 6.41, we plot the nonlinear residual for group one and group two. This figure demonstrates that the nonlinear residual is strongest near material interfaces and where the steepest gradients occur. By considering Figures 6.37, 6.40 and 6.41 simultaneously, we can see the interaction between material interfaces, the scalar flux, and the nonlinear residual.

We theorize that the stall in the nonlinear residual is due to the truncation error of the low-order problem. To test this, we'll solve the problem on a series of meshes and consider the nonlinear residual on each of these meshes. Instead of plotting the $\infty-$norm of the nonlinear residual, we'll compare the scaled $1-$norm, $h^2\|\cdot\|_1$. These results are displayed in Figure 6.42. This figure demonstrates that with these meshes, by the end of the iteration the Monte Carlo

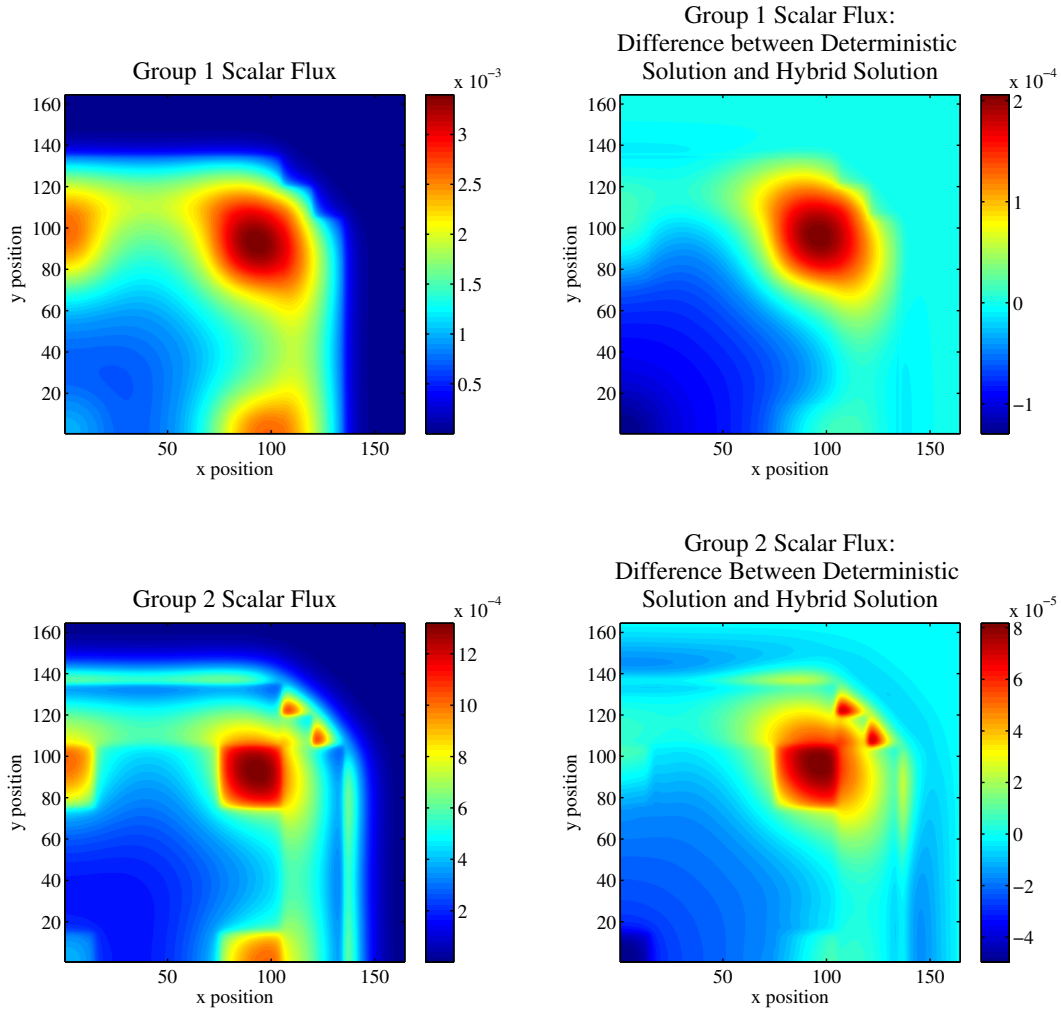

Figure 6.34: On the left, we plot the group one and group two eigenvector, respectively. On the right, we see the difference between the eigenvector computed using hybrid methods and deterministic methods.

error is lower than the truncation error of the low-order problem and the nonlinear residual stalls at the level of the truncation error.

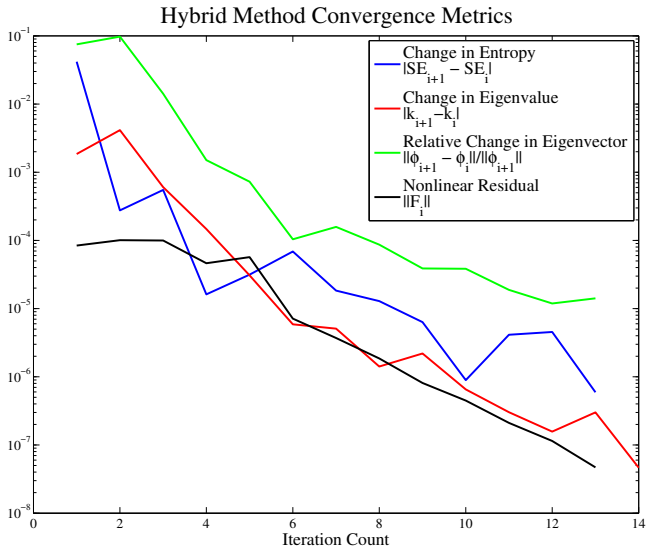

Figure 6.35: We plot the four convergence metrics for the first simulation for the LRA-BWR problem.

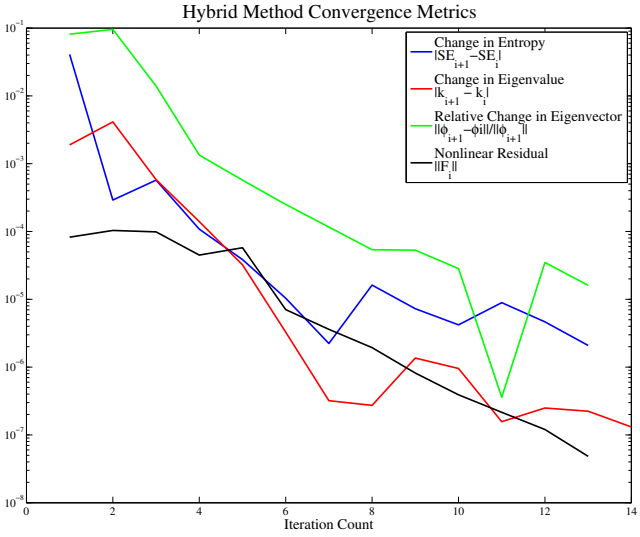

Figure 6.36: We plot the four convergence metrics for the second simulation for the LRA-BWR problem.

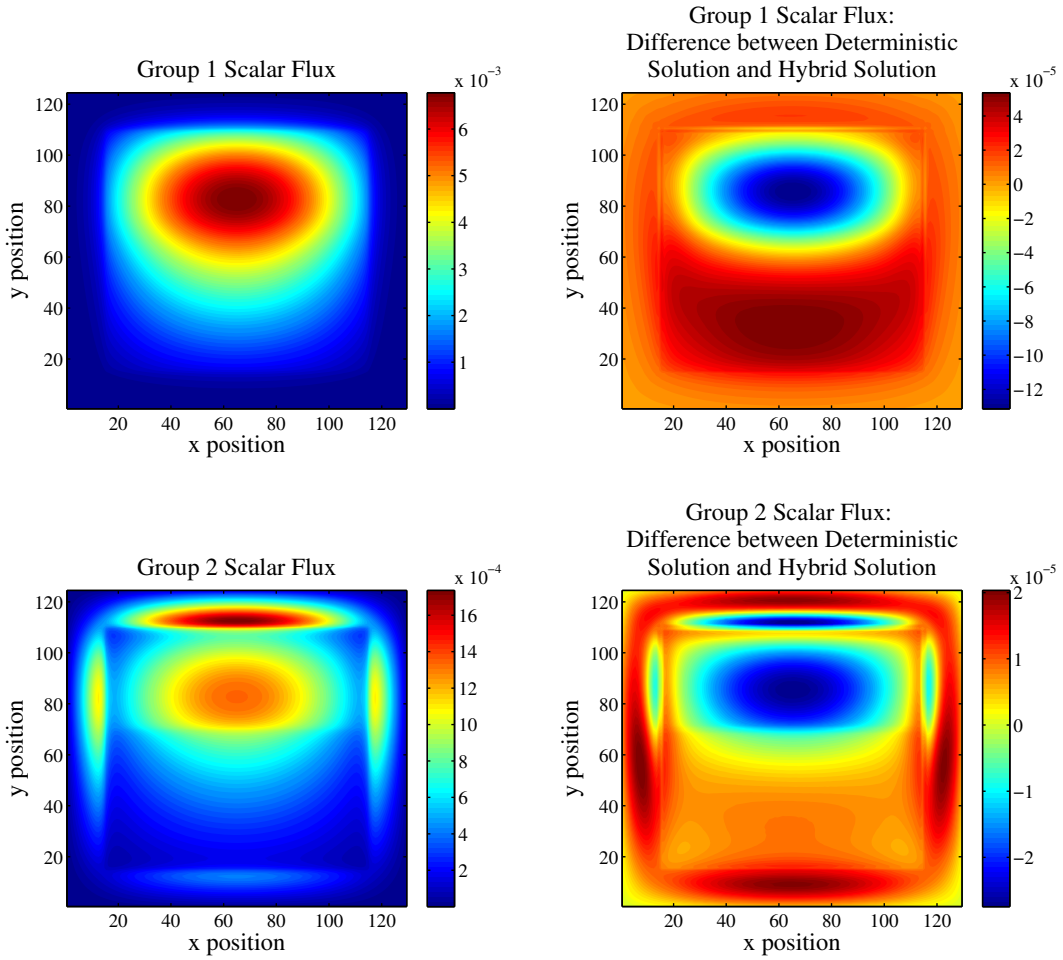

Figure 6.37: On the left, we plot the group one and group two eigenvector, respectively. On the right, we see the difference between the eigenvector computed using hybrid methods and deterministic methods.

Zion Reactor

We have run two independent NDA-NCA(MC) simulations and compiled results. After 12 outer iterations and maximum of 2.048 $\times 10^{12}$ particles per function evaluation and using a 136 $\times$ 136, we have locked onto the first six digits of the eigenvalue

$$\begin{aligned} k_{eff,1} &= \mathbf{1.274762}569404242 \\ k_{eff,2} &= \mathbf{1.274762}670387974 \end{aligned}$$

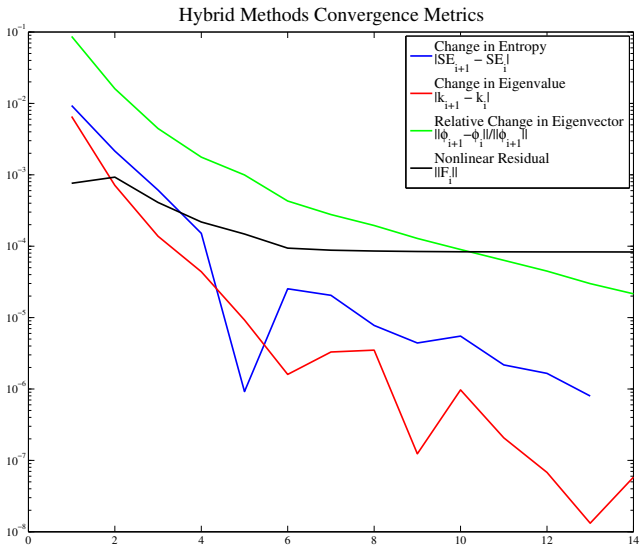

Figure 6.38: We plot the four convergence metrics for the first simulation for the TMR problem.

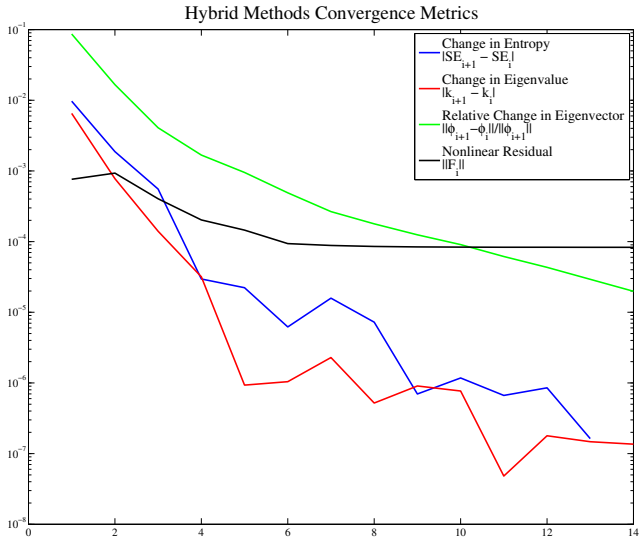

Figure 6.39: We plot the four convergence metrics for the second simulation for the TMR problem.

in both simulations. In Figure 6.43, we plot the eigenvector computed using the hybrid method alongside the difference between the eigenvector computed using hybrid methods and deterministic methods.

In Figure 6.44, we look at a combination of the four convergence metrics for each simulation. Just like the TMR problem, it appears that the nonlinear residual is stalling, however the eigenvector still appears to be converging and both the eigenvalue and the Shannon entropy are converging. Furthermore, both the eigenvalue and eigenvector compare well with deterministic solutions.

Like we did for the TMR reactor problem, we'll consider the spatial dependence of the nonlinear residual. In Figure 6.45, we plot the nonlinear residual for group one and group two. This figure demonstrates that the nonlinear residual is strongest near material interfaces and where the steepest gradients occur. By considering Figures 6.43, 6.45 and 6.46 we can see that the behavior is similar to the behavior for the TMR problem.

6.6 Conclusions

Throughout this chapter, we have demonstrated that it is possible to build hybrid methods by introducing a Monte Carlo transport sweep in place of a deterministic transport sweep in any of the moment-based algorithms which we have considered throughout this thesis. By astutely managing the increase of particles, we can "track" r-linear convergence with JFNK-NDA(MC).

In many ways, computing the eigenvalue is easier than the fixed-source problem. This is because in our computations, the error (or relative residual) in the eigenvector is usually 2 - 3 orders of magnitude larger than the error (or relative residual) in the eigenvalue. Therefore, in order to get 5 digits correct in the eigenvalue, we may only need to compute the eigenvector accurately to 2 or 3 digits.

Of course, the true goal with these algorithms is to be able to accurately compute the dominant eigenvalue for realistic reactor problems. We have demonstrated that these algorithms work on two large, 2-D, 2-group eigenvalue problems. NDA-NCA is an ideal algorithm for hybrid computations of the k-eigenvalue as much of the work is moved away from high-order transport sweeps to low-order eigenvalue solves. These low-order solves take place in the absence of Monte Carlo noise and therefore can achieve results without the headaches we encounter when trying to implement JFNK-NDA(MC).

Furthermore, by using an efficient parallel implementation of the Monte Carlo transport sweep, these algorithms become very practical and can yield more accurate solutions than deterministic methods. By ridding the high-order solution of the angular and spatial discretizations, we can compute more physically accurate neutron fluxes which lead to more accurate eigenvalues. Figures 6.34 and 6.37 demonstrate that the hybrid methods can achieve higher

accuracy in regions where the scalar flux is rapidly changing, such as near material interfaces, or areas where the scalar flux (and fission source) is strongest.

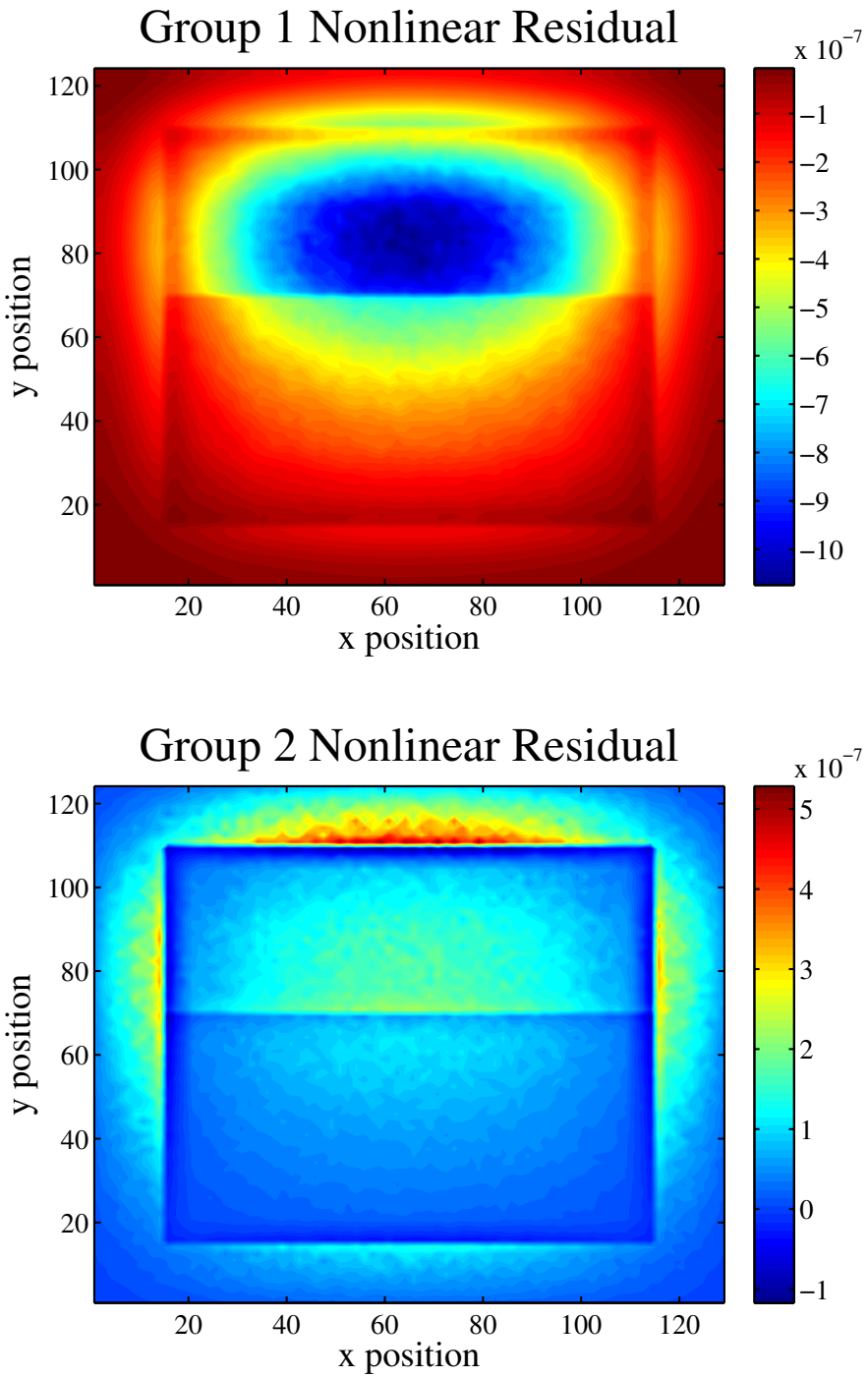

Figure 6.40: We plot the nonlinear residual as a function of space for a 75 × 78 mesh.

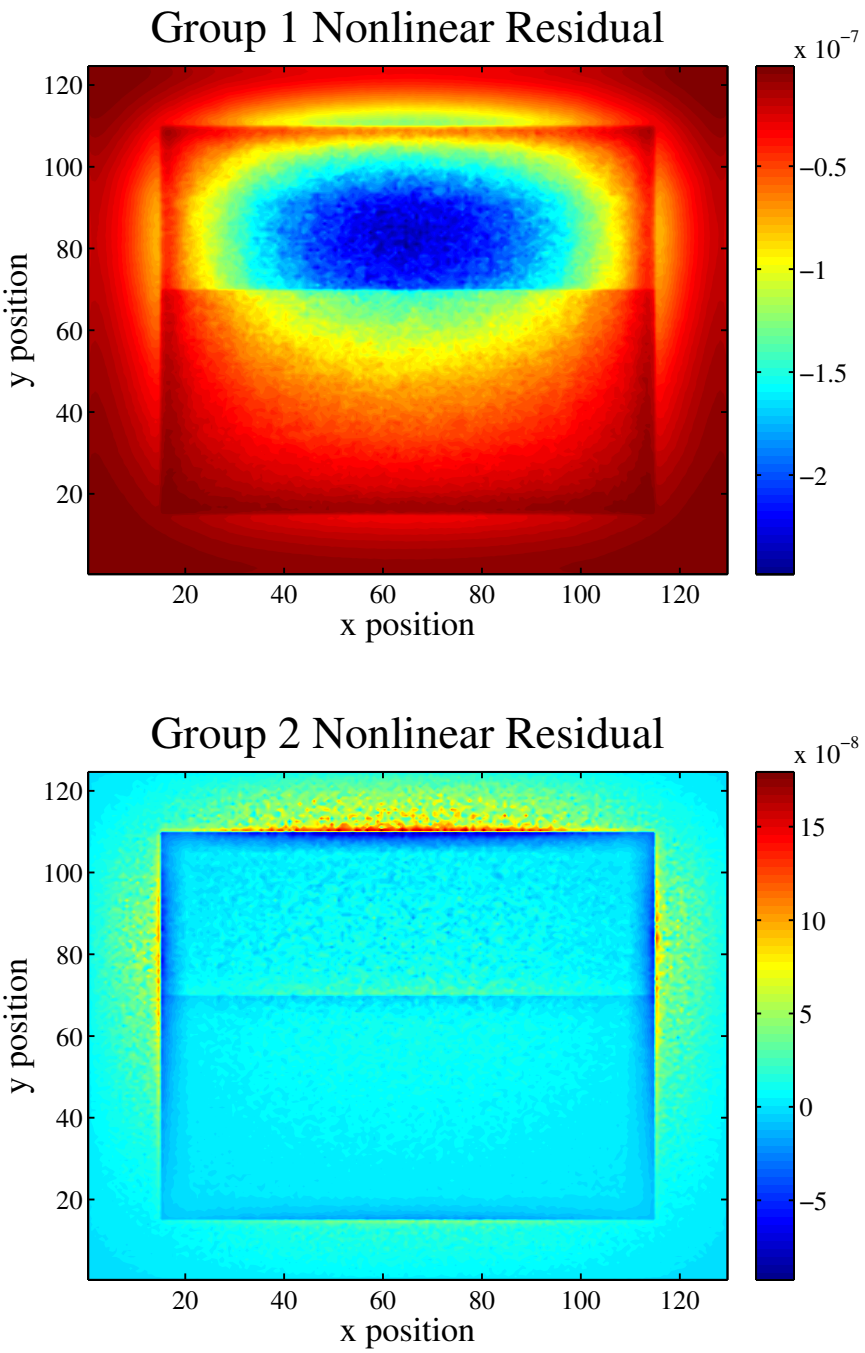

Figure 6.41: We plot the nonlinear residual as a function of space for a 175 × 182 mesh.

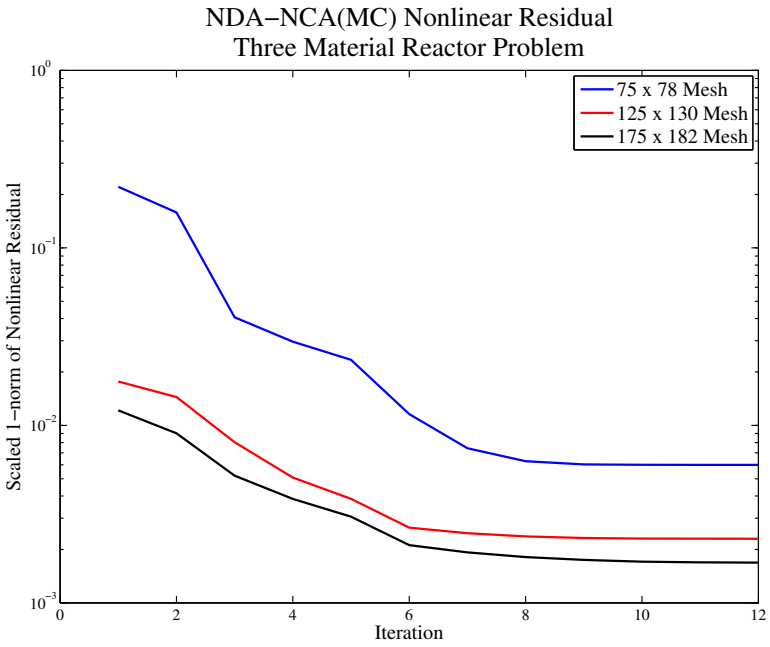

Figure 6.42: We plot the scaled one-norm of the nonlinear residual by iteration for a series of three meshes.

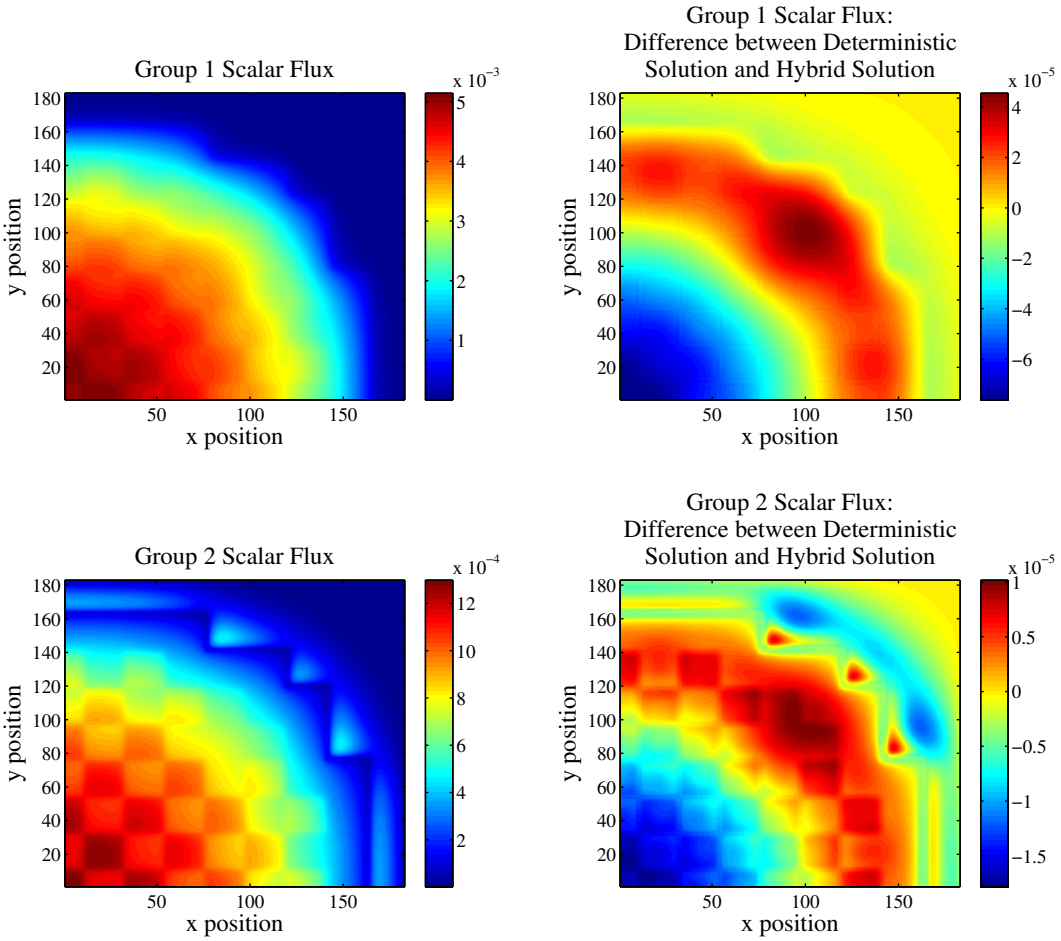

Figure 6.43: On the left, we plot the group one and group two eigenvector, respectively. On the right, we see the difference between the eigenvector computed using hybrid methods and deterministic methods.

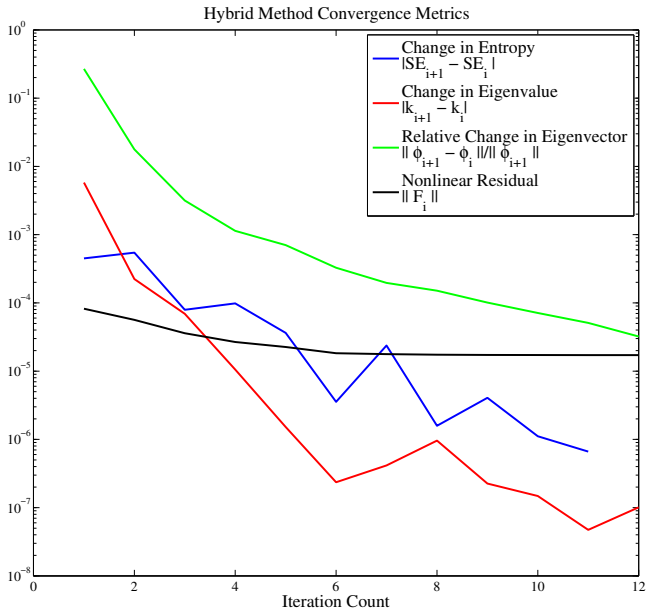

Figure 6.44: We plot the four convergence metrics for the Zion reactor problem.

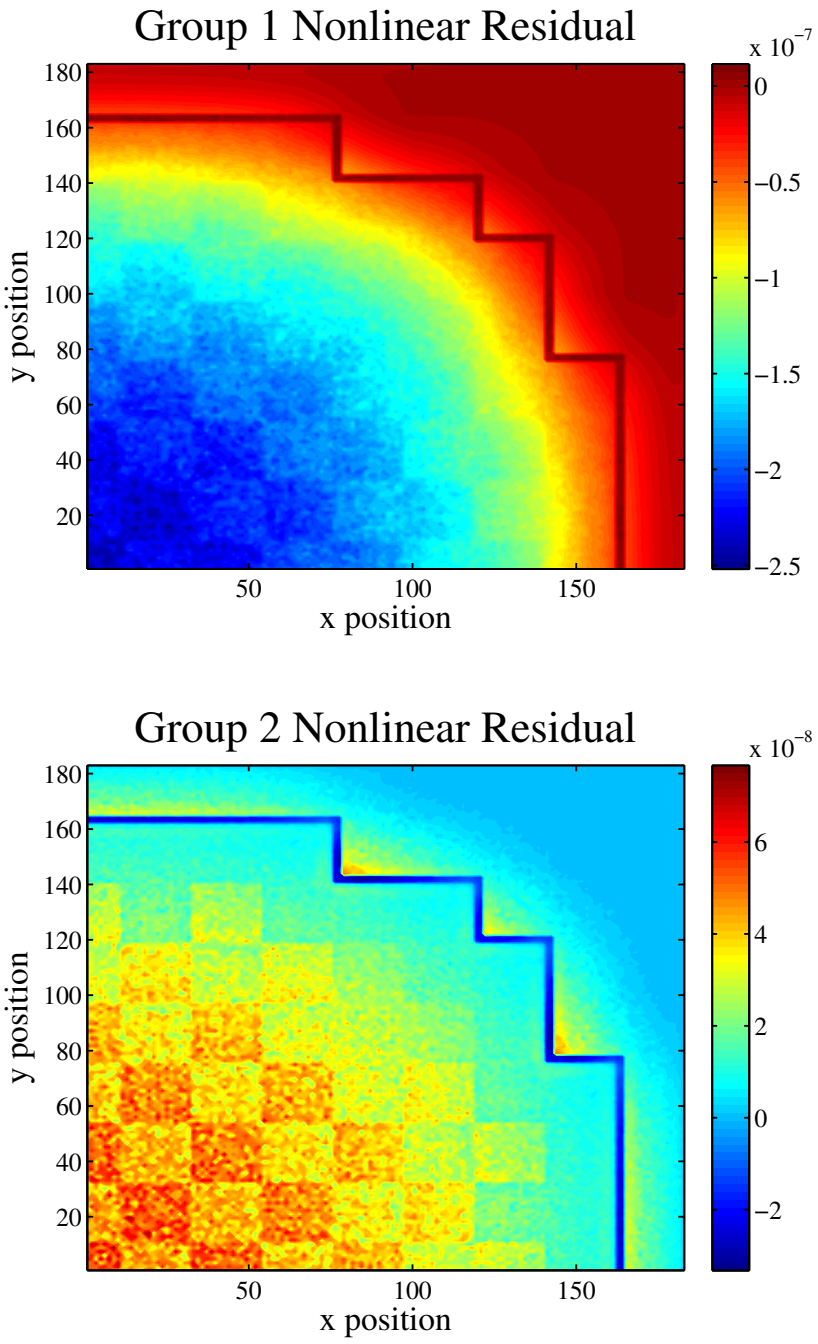

Figure 6.45: We plot the nonlinear residual as a function of space for a 136 × 136 mesh.

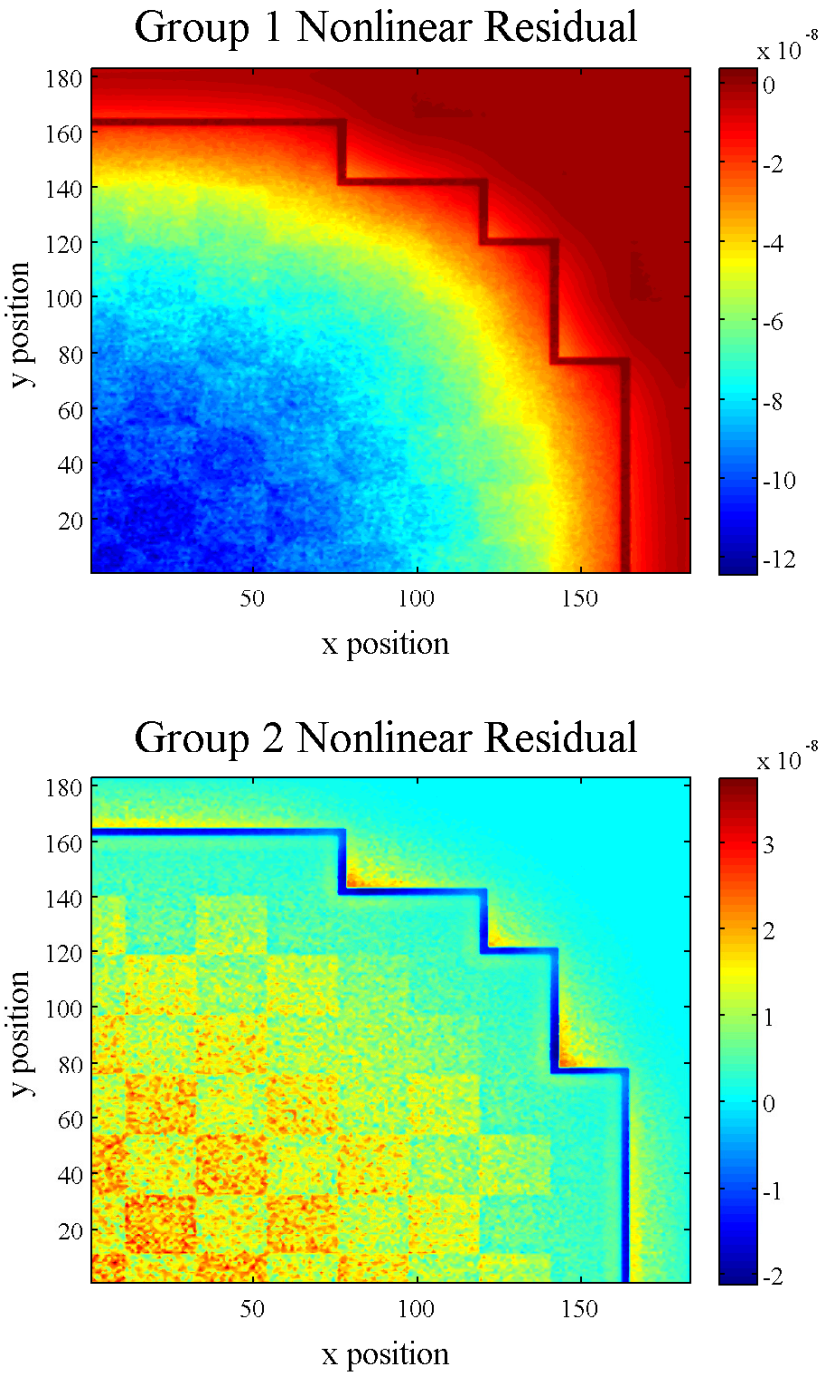

Figure 6.46: We plot the nonlinear residual as a function of space for a 204 $\times$ 204 mesh.

Chapter 7

Conclusion

Throughout this thesis we have considered numerical methods for solving the neutron transport fixed-source problem, as well as the neutron transport k-eigenvalue problem. We built these algorithms by first considering the most simple case - one-dimensional, one energy group, homogenous medium problems using deterministic methods. In this setting we considered the fixed-source problem first and looked at the most straightforward method, source iteration. From source iteration we showed how we could use angular moments of the transport equation to increase the convergence rate [18]. At this point we took our findings in one space dimension and applied these methods to the two-dimensional fixed-source problem. It was straightforward to demonstrate the effectiveness of these algorithms in higher dimension.

After concluding our work on deterministic methods for the fixed-source problem, we turned our attention to solving the k-eigenvalue problem. Again we began by implementing a simple power iteration and showing how ineffective this algorithm can be, even for simple one-dimensional problems. We then accelerated the convergence of the eigenvalue problem by turning to moment-based acceleration. This allowed us to move the majority of the computational work to a low-order problem that exists only in a reduced phase-space. As in [23], we used Newton's method to solve the low-order problem, however we showed that we could significantly decrease the cost of the low-order eigenvalue solving by exploiting the nature of the Jacobian matrix. We demonstrated that these algorithms work very well on both a series of simple one-dimensional, one group test problems, as well as some much more sophisticated two-dimensional test problems.

After completing a study of deterministic methods for solving the fixed-source and k-eigenvalue problem, we segued into the heart of the thesis by considering stochastic methods for neutron transport. These methods utilize Monte Carlo algorithms to simulate each physical process which takes place between the neutrons and underlying medium. This chapter allowed us to develop terminology and machinery to be used in our discussion of hybrid methods. At

the end of our discussion of stochastic methods we demonstrated the increased effectiveness of the continuous energy deposition [10] tallies which were critical for the effective implementation of hybrid methods.

In the following chapter, we discussed hybrid methods for neutron transport. These methods combine the accelerators we developed for deterministic methods and Monte Carlo simulation. We used Monte Carlo to approximate the transport sweep process and used the outcome of these stochastic transport sweeps to compute closure relationships for low-order problems which allowed us to accelerate convergence.

We began our discussion of hybrid methods with a long, in-depth study which demonstrated the care which must be taken when implementing a hybrid method, even for a simple one-dimensional, one group, homogeneous medium test problem [32]. In this section, we learned about the importance of analytic Jacobian-vector products which allow us to bypass noise-ridden finite-difference computations. We also completed a detailed study which analyzed the behavior of Newton's method when the function and Jacobian-vector product evaluations contained stochastic noise. By managing the number of particle histories per function evaluation astutely, we demonstrated that we could achieve r-linear convergence [30].

These one-dimensional fixed-source findings led to a straight-forward implementation of the NDA-NCA(MC) hybrid algorithm for computing the k-eigenvalue. Using similar particle management techniques, we found that we were able to efficiently compute the dominant eigenvalue using these hybrid methods.

At this point, we moved on to two-dimensional hybrid test problems. Before directly jumping into implementing these algorithms in two-dimensions, we spent some time developing a highly-parallel Monte Carlo transport sweep implemented in `C++` using OpenMP and MPI. We demonstrated the strong scaling efficiency of this algorithm for realistic domain and material properties [34].

Lastly, we implemented JFNK-NDA(MC) and NDA-NCA(MC) for two-dimensional, two group, multi-material test problems. Using our one-dimensional findings, these implementations were straight-forward, as well. For JFNK-NDA(MC), we found that we could track r-linear convergence using the ideas we outlined in [30]. NDA-NCA(MC) proved to be a success, as well. Using the same spatial mesh, the deterministic method provided a results demonstrating that the LRA-BWR reactor [1, 28] was subcritical, however the hybrid method correctly showed that the reactor was, in fact, supercritical. Thus, on a given mesh, the hybrid method provides more physically accurate approximations to the eigenvalue. For the two other two-dimensional k-eigenvalue test problems, we were able to see the interaction between the low-order truncation error and the Monte Carlo error. This interaction manifested itself in the stagnation of the nonlinear residual for the low-order problem. We showed that by refining the spatial mesh we could reduce the low-order truncation error and drive the nonlinear residual to a tighter

tolerance.

By starting with simple deterministic problems and gaining a very detailed understanding of the algorithms, we were able to successfully build hybrid methods for solving very challenging two-dimensional eigenvalue problems. Furthermore, we were able to demonstrate that these hybrid methods could produce more physically accurate solutions which was the ultimate goal of this research.

7.1 Future Work

Throughout the development of this thesis we have come across a few research directions that, while outside the scope of this dissertation, would be very interesting to spend time rigorously investigating. These research directions include

1. Implementing these algorithms in three-spatial dimensions.
2. Implementing these ideas for a time-dependent fixed-source problem.
3. Coupling these hybrid methods with other physics models through the low-order problem.
4. Considering algorithm adjustments which would allow for the continuous treatment of energy in the high-order problem.

We'll briefly comment on each directions.

7.1.1 Implementation in Three-Spatial Dimensions

The idea here is very straight-forward. Using the knowledge gained through the development of these hybrid algorithms in one and two-space dimensions, implement JFNK-NDA(MC) and NDA-NCA(MC) in three-spatial dimensions. These algorithms will needed to be coded entirely in an environment outside of MATLAB as the two-dimensional implementations in which the low-order problem is solved in MATLAB has pushed this environment to its limits. Following a similar programming model, an efficient, scalable parallel three-dimensional transport sweep could be developed. The solution to the low-order problem (fixed-source or eigenvalue) requires Newton's method. Implementations of Newton's method in C++ already exist and could be leveraged for an efficient three-dimensional implementation.

7.1.2 Implementation of NDA(MC) for a Time-Dependent Fixed-Source Problem and Multi-Physics Coupling

We'll discuss these two ideas together as they are intimately related. In reality, material cross-sections can be temperature dependent and therefore dependent on the scalar flux as

it changes throughout time [21, 8]. An investigation into how these hybrid methods work for time-dependent problems could produce interesting results. Furthermore, an implementation in which the neutronics are coupled to other physics models would likely take place through the low-order system. This offers further incentive to use a moment-based acceleration scheme. These other physics models could likely interact only in the reduced phase space, without interacting with the seven-dimensional angular flux.

7.1.3 Continuous Energy in the High-Order Problem

One of the most significant challenges in using the multi-group formulation of the neutron transport equation is the development of multi-group cross-sections that preserve total, scattering, and fission reaction rates. By treating energy continuously in the high-order problem, there is the potential to develop more accurate material cross-sections during the iterative solution process. This would require saving energy distribution functions for the high-order problem in order to accurately model the neutron scattering process (this does not come into play with the fission process since the fission spectrum is known). This may lead to memory concerns if we are required to store these energy distribution functions everywhere in space, however the potential advantages of using a continuous treatment of energy in the high-order problem make this a worthwhile research direction.

REFERENCES

[1] Argonne code center: Benchmark problem book, 1977. Prepared by the Computational Benchmark Problems Committee of the Mathematics and Computational Division of the Americal Nuclear Society, Supplement 2.

[2] Marvin L. Adams and Edward W. Larsen. Fast iterative methods for discrete-ordinates particle transport calculations. *Progress in Nuclear Energy*, 40(1):3–159, 2002.

[3] F.B. Brown. On the use of Shannon entropy of the fission distribution for assessing convergence of Monte Carlo criticality calculations. In *PHYSOR*, Vancouver, BC, Canada, 2006. ANS Topical Meeting on Reactor Physics Organized and Hosted by the Canadian Nuclear Society.

[4] Matthew T. Calef, Erin D. Fichtl, James S. Warsa, Markus Berndt, and Neil N. Carlson. Nonlinear Krylov acceleration applied to a discrete ordinates formulation of the k-eigenvalue problem. *Journal of Computational Physics*, 238(0):188 – 209, 2013.

[5] L.L. Carter and E.D. Cashwell. Particle-transport simulation with the Monte Carlo method: Prepared for the division of military application, U.S. Energy Research and Development Administration. Technical report, Technical Information Center, Office of Public Affairs, U.S. Energy Research and Development Administration, 1975.

[6] S. Chandrasekhar. *Radiative Transfer*. Dover Publications, Inc, New York, 1960.

[7] James J. Duderstadt and Louis J. Hamilton. *Nuclear Reactor Analysis*. John Wiley & Sons, Inc., New York, 1976.

[8] James J. Duderstadt and William R. Martin. *Transport Theory*. John Wiley & Sons, Inc., New York, 1979.

[9] H. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra Applications*, 16:197–221, 2009.

[10] J. Fleck and J. Cummings. Implicit Monte Carlo scheme for calculating time and frequency dependent non-linear radiation transport. *Journal of Computational Physics*, 8:313–342, 1971.

[11] Daniel F. Gill and Yousry Y. Azmy. Newton's method for solving k-eigenvalue problems in neutron diffusion theory. *Nuclear Science and Engineering*, 167:141–153, 2011.

[12] Daniel F. Gill, Yousry Y. Azmy, J.S. Warsa, and J.D. Densmore. Newton's method for the computation of k-eigenvalue problems in S_N transport applications. *Nuclear Science and Engineering*, 168:37–58, 2011.

[13] Mark A. Goffin, Christopher M.J. Baker, Andrew G. Buchan, Christopher C. Pain, Matthew D. Eaton, and Paul N. Smith. Minimising the error in eigenvalue calculations involving the Boltzmann transport equation using goal-based adaptivity on unstructured meshes. *Journal of Computational Physics*, To appear in 2013.

[14] V.Ya. Gol'din. A Quasi-Diffusion method of solving the kinetic equation. *USSR Computational Mathematics and Mathematical Physics*, 4(6):136–149, 1964.

[15] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, 1995.

[16] C.T. Kelley. Multilevel source iteration accelerators for the linear transport equation in slab geometry. *Transport Theory and Statistical Physics*, 24(4):679–707, 1995.

[17] D. A. Knoll, H. Park, and C.K. Newman. Acceleration of k-eigenvalue/criticality calculations using the Jacobian-free Newton-Krylov method. *Nuclear Science and Engineering*, 167(2):133–140, February 2011.

[18] D. A. Knoll, Kord Smith, and H. Park. Application of the Jacobian-free Newton-Krylov method to nonlinear acceleration of transport source iteration in slab geometry. *Nuclear Science and Engineering*, 167(2):122–132, February 2011.

[19] K. Lanthrop. Spatial differencing of the transport equation: Positivity vs. accuracy. *Journal of Computational Physics*, 4:475, 1969.

[20] E.W. Larsen and J. Yang. A Functional Monte Carlo Method for k-Eigenvalue Problems. *Nuclear Science and Engineering*, 159(2):107–126, February 2008.

[21] E.E. Lewis and W.F. Miller. *Computational Methods of Neutron Transport*. American Nuclear Society, Inc., La Grange Park, 1993.

[22] S. Nakamura. *Computational Methods in Engineering and Science with Applications to Fluid Dynamics and Nuclear Systems*. Robert E. Krieger Publishing Company, Malabar, 1986.

[23] H. Park, D. A. Knoll, and C.K. Newman. Nonlinear acceleration of transport criticality problems. *Nuclear Science and Engineering*, 172(1):52–65, September 2012.

[24] F. A. Potra and H. Engler. A characterization of the behavior of the Anderson acceleration on linear problems. *To appear in Linear Algebra and its Applications*.

[25] Paul K. Romano and Benoit Forget. The OpenMC Monte Carlo particle transport code. *Annals of Nuclear Energy*, 51:274–281, 2013.

[26] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

[27] Y. Saad and M. Schultz. GMRES a generalized minimal residual algorithm for solving nonsymmetric linear systems. *Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[28] Kord. S. Smith. An analytic nodal method for solving the two-group, multidimensional static and transient neutron diffusion equations, 1979. Masters Thesis, Massachusetts Institute of Technology.

[29] H. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

[30] Jeffrey A. Willert, Xiaojun Chen, and C.T. Kelley. Newton's method for Monte Carlo-based residuals. *Submmitted for Publication*, 2013.

[31] Jeffrey A. Willert and C.T. Kelley. Efficient solutions to the NDA-NCA low-order eigenvalue problem. In *M&C 2013 Proceedings*, pages 1934–1941, Sun Valley, ID, May 5 - 9, 2013 2013. American Nuclear Society.

[32] Jeffrey A. Willert, C.T. Kelley, D. A. Knoll, and H. Park. Hybrid methods for neutron transport. *Submitted for Publication*, 2012.

[33] Jeffrey A. Willert, C.T. Kelley, D. A. Knoll, and H. Park. A hybrid approach to the neutron transport k-eigenvalue problem using NDA-based algorithms. In *M&C 2013 Proceedings*, pages 2725–2735, Sun Valley, ID, May 5 - 9, 2013 2013. American Nuclear Society.

[34] Jeffrey A. Willert, C.T. Kelley, D. A. Knoll, and H. Park. Scalable hybrid deterministic/Monte Carlo neutronics simulations in two space dimensions. *Submitted for Publication*, 2013.

[35] Sunghwan Yun and Nam Zin Cho. Acceleration of source convergence in Monte Carlo k-eigenvalue problems via anchoring with a p-CMFD deterministic method. *Annals of Nuclear Energy*, 37:1649–1658, 2010.

APPENDICES

Appendix A

Linear and Nonlinear Iterative Solvers

This appendix is devoted to describing the various linear and nonlinear solvers that were used in this thesis. The goal here is to provide both a description of how these methods work and are implemented and to provide some theory describing convergence. We assume that the reader is familiar with basic linear algebra, calculus and matrix theory.

A.1 Linear Iterative Methods

The goal of linear iterative methods are to solve problems of the form

$$Ax = b,$$

where A is an $n \times n$ matrix and x and b are $n \times 1$ vectors. For the purposes of this thesis, it is only necessary to consider $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$.

In the remainder of this appendix, uppercase letters will denote matrices and lower case letters will denote vectors. If we need to refer to a specific element of a vector, we will subscript using the row number. For example, x_j refers to the j^{th} component of the vector x. When referring to elements of a matrix, we will subscript using the row number followed by the column number, as is standard. That is, $a_{i,j}$ (or a_{ij}) refers to the element in the i^{th} row and j^{th} column of the matrix A.

For many of the applications in this thesis we will find that we never build actual representations of the matrix A. Instead, we will write functions that take in a vector, x, and return a matrix-vector product, Ax. For this reason (among others), we avoid using direct methods and concentrate our focus on linear iterative methods. When executing a transport sweep, as

in Chapter 2, computing the entries in A is not feasible. Instead, we can easily compute Ax. In general, we assume that A is not symmetric and that we do not have an easy method for computing the transpose of A, A^T.

A.1.1 Matrix Preliminaries

For many of our algorithms, we must be able to compute a residual, r, and error, e. We compute these using the following formulas,

$$r = b - Ax \text{ and} \tag{A.1}$$

$$e = x - x^*, \tag{A.2}$$

where $x^* = A^{-1}b$, the exact solution. It is easy to see that

$$r = b - Ax = A(A^{-1}b - x) = A(x^* - x) = -Ae. \tag{A.3}$$

In the following analyses we'll often take advantage of matrix norms [15]:

Definition 5 (Matrix Norm)**.** *Let $\|\cdot\|$ denote a vector norm on $\mathbb{R}^n$. We define the induced matrix norm of a square $n \times n$ matrix A defined by*

$$\|A\| = \max_{x \in \mathbb{R}^n} \frac{\|Ax\|}{\|x\|} \tag{A.4}$$

or, equivalently

$$\|A\| = \max_{\|x\|=1} \|Ax\|. \tag{A.5}$$

Furthermore, we'll need to discuss *condition numbers* when analyzing convergence properties of Krylov methods.

Definition 6 (Condtion Number)**.** *The condition number of a matrix A, $\kappa(A)$, with the norm $\|\cdot\|$ is given by*

$$\kappa(A) = \|A\|\|A^{-1}\|. \tag{A.6}$$

Finally, we must utilize the notion of *eigenvalues* in the analysis of iterative methods in the following sections.

Definition 7 (Eigenvalue, Eigenvector, Spectrum, Spectral Radius)**.** *We say that λ is an*

eigenvalue of the matrix A if there exists a nonzero vector x such that

$$Ax = \lambda x. \tag{A.7}$$

We say that x is the eigenvector corresponding to the eigenvalue λ. Together, we'll refer to the pair (λ, x) as an eigenpair.

The set of eigenvalues of matrix A, denoted $\sigma(A)$, is known as the spectrum of the matrix A. Furthermore, the spectral radius of the matrix A, $\rho(A)$ is given by

$$\rho(A) = \max_{\lambda \in \sigma(A)} |\lambda|. \tag{A.8}$$

Our understanding of induced matrix norms and the previous definition lead to the following lemma [15].

Lemma 1. *For any choice of induced matrix norm, we have*

$$\rho(A) \leq \|A\| \tag{A.9}$$

Proof. Consider an eigenpair (λ, x). Then,

$$\begin{aligned} Ax &= \lambda x \\ \|Ax\| &= |\lambda| \|x\| \end{aligned}$$

Rearranging terms and utilizing the definition of matrix norms yields

$$\begin{aligned} |\lambda| \|x\| &\leq \|A\| \|x\| \\ |\lambda| &\leq \|A\| \end{aligned}$$

for all λ. □

A.1.2 Stationary Iterative Methods

The most simple type of linear iterative methods is called a stationary iterative method, where we define an iteration matrix that takes us from one iterate to the next. If we decompose A as

$A = B - C$ where B is nonsingular, this can be realized as:

$$\begin{aligned}(B-C)x &= b && \text{(A.10)}\\ Bx &= Cx + b && \text{(A.11)}\\ x &= B^{-1}Cx + B^{-1}b && \text{(A.12)}\\ x &= Mx + d && \text{(A.13)}\end{aligned}$$

where $M = B^{-1}C$.

Now, we can iterate as follows

$$x_{n+1} = Mx_n + d \tag{A.14}$$

where (we hope) $x_n \to x^*$ as $n \to \infty$ where $Ax^* = b$. For this iterative scheme, we have the following theorem describing convergence [15].

Theorem 4. *If $\|M\| < 1$, then $(I - M)$ is nonsingular and the iteration A.14 converges to $(I - M)^{-1}d$.*

Proof. Suppose $\|M\| < 1$. We will show that

$$\sum_{i=0}^{\infty} M^i = (I - M)^{-1}.$$

To do so, let us consider the following partial sum

$$S_k = \sum_{i=0}^{k} M^i.$$

It is easy to see that these partial sums $\{S_k\}_{k=1}^{\infty}$ form a Cauchy sequence in $\mathbb{R}^{n\times n}$. For all $j > k$, consider

$$\begin{aligned}\|S_k - S_j\| &= \Big\| \sum_{i=k+1}^{j} M^i \Big\| \\ &\leq \sum_{i=k+1}^{j} \|M^i\| \\ &\leq \sum_{i=k+1}^{j} \|M\|^i \\ &= \|M\|^{k+1} \left(\frac{1 - \|M\|^{m-k}}{1 - \|M\|} \right)\end{aligned}$$

where the last equality is a consequence of the sum of a finite geometric series. Since $\|M\| < 1$, $\|S_k - S_j\| \to 0$ as $k \to \infty$.

Therefore, $S_k \to S$ for some $S \in \mathbb{R}^{n \times n}$. Furthermore, we have

$$S = MS + I$$

by definition of S. Thus,

$$\begin{aligned} S - MS &= I && \text{(A.15)} \\ S(I - M) &= I && \text{(A.16)} \\ S &= (I - M)^{-1} && \text{(A.17)} \end{aligned}$$

as desired.

Now, for any x_0, we can compute x_n using simple substitution,

$$x_{n+1} = M^{n+1}x_0 + \sum_{i=0}^{n} M^i d.$$

Therefore, $x_n \to Sd$ as $n \to \infty$. Since $S = (I - M)^{-1}$, we have $x_n \to (I - M)^{-1}d$. □

We'd like to terminate these methods when the error, $e = x - x^*$, is small, but we cannot compute the error without knowing the true solution. Instead, we must terminate when the residual, $r = b - Ax$ is small. For this reason, it is important to be able to relate the error and the linear residual. To do so, we employ the following theorem [15].

Theorem 5. *Let $x_0 \in \mathbb{R}^n$ be the initial iterate. Assume A is nonsingular and let $x^* = A^{-1}b$. Then for all $x \in \mathbb{R}^n$, we have*

$$\frac{\|e\|}{\|e_0\|} \leq \kappa(A)\frac{\|r\|}{\|r_0\|}$$

Theorem 5 ensures that if the condition number of A is not too large, small residuals indicate small errors.

While there are several more intricate stationary iterative methods, we can improve convergence by considering a class of linear iterative methods called Krylov methods.

A.1.3 Krylov Methods

Krylov methods are a class of non-stationary iterative methods. Krylov methods minimize some measure of the error (norm of the error, norm of the residual, etc.) over the shifted k^{th}

Krylov subspace,

$$x_0 + \mathcal{K}_k,$$

in which x_0 is the initial iterate and $\mathcal{K}_k$ is the k^{th} Krylov subspace,

$$\mathcal{K}_k = \text{span}(r_0, Ar_0, \ldots, A^{k-1}r_0)$$

for $k \geq 1$, where $r_0 = b - Ax_0$ [15, 27].

We'll assume for the remainder of this section that A is a nonsingular $n \times n$ matrix and let x^* denote the solution to the linear system $Ax = b$. Furthermore, we'll let $\{r_k\}_{k\geq 0}$ denote the sequence of residuals built from the sequence of iterates $\{x_k\}_{k\geq 0}$, where $r_k = b - Ax_k$.

While there are several Krylov methods suitable for solving linear problems (see [15]), our focus will remain primarily on GMRES, or the Generalized Minimal RESidual algorithm. GMRES allows us to solve non-symmetric problems, unlike Conjugate Gradient iteration, and has a very strong, well-understood theory. Furthermore, GMRES does not require us to build and store a representation of the matrix, nor does it require the computation of A^T [15, 27]. For these reasons, GMRES is well suited for our current application.

At the k^{th} iteration of GMRES, we solve the least squares problem

$$\min_{x \in x_0 + \mathcal{K}_k} \|b - Ax\|_2. \tag{A.18}$$

That is, at the k^{th} iteration, we minimize the 2-norm of the residual over the shifted k^{th} Krylov subspace.

The following analysis of GMRES will follow the method put forth in [15]. For each $x \in x_0 + \mathcal{K}_k$ we can represent x as x_0 plus a linear combination of basis elements of the k^{th} Krylov subspace,

$$x = x_0 + \sum_{i=0}^{k-1} \alpha_i A^i r_0.$$

Therefore, the residual can be represented in the following manner,

$$\begin{aligned} r &= b - Ax \\ &= b - Ax_0 - A\sum_{i=0}^{k-1} \alpha_i A^i r_0 \\ &= r_0 - \sum_{i=0}^{k-1} \alpha_i A^{i+1} r_0 \\ &= r_0 - \sum_{i=1}^{k} \alpha_{i-1} A^i r_0 \\ &= p(A) r_0 \end{aligned}$$

where $p(z) = 1 - \sum_{i=1}^{k} \alpha_{i-1} z^i$. We'll define $\mathcal{P}$ as the set of k^{th} degree residual polynomials, that is, the set of polynomials $p(z)$ such that the degree of p is k and $p(0) = 1$. Since x_k satisfies Eq. A.18, we have the following theorem.

Theorem 6. *Suppose A is a nonsingular matrix and let x_k be the k^{th} GMRES iterate, satisfying Eq. A.18. Then, for all $p_k \in \mathcal{P}_k$, we have,*

$$\|r_k\|_2 = \min_{p \in \mathcal{P}_k} \|p(A) r_0\|_2 \leq \|p_k(A) r_0\|_2.$$

Theorem 6 leads directly to the following corollary.

Corollary 1. *Let A be a nonsingular matrix and let x_k be the k^{th} GMRES iterate, satisfying Eq. A.18. Then for all $p_k \in \mathcal{P}_k$, we have*

$$\|r_k\|_2 \leq \|p_k(A)\|_2 \|r_0\|_2,$$

or

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \|p_k(A)\|_2. \tag{A.19}$$

Equation A.19 grants us a method for analyzing the convergence of GMRES based upon the spectrum of the matrix A. Furthermore, it allows us to prove that GMRES must converge in at most n iterations when A is an $n \times n$ matrix.

Theorem 7. *Let A be an $n \times n$ nonsingular matrix. Then GMRES will converge to the exact solution in at most n iterations.*

Proof. Let $p(z) = \det(A - zI)$. By definition, p has degree n and $p(0) \neq 0$, since A is nonsingular.

Then

$$\hat{p}(z) = \frac{p(z)}{p(0)}$$

is a residual polynomial. Furthermore, since every matrix A satisfies its characteristic equation, $\hat{p}(A) = 0$. Therefore, by Equation A.19,

$$\|r_n\|_2 \leq 0,$$

which implies r_n must, in fact, be zero and thus $x_n = x^*$. □

In the case in which A is diagonalizable, Equation A.19 can be simplified [15, 27].

Theorem 8. *Suppose A is diagonalizable with nonsingular diagonalizing transform V (i.e. $A = V^{-1}\Lambda V$). Then for all $p \in \mathcal{P}_k$,*

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa(V) \max_{\lambda \in \sigma(A)} |p(\lambda)|. \tag{A.20}$$

A.1.4 Other Krylov Methods

While GMRES is an excellent method for solving non-symmetric linear systems, it may not always be a viable option. Within the GMRES algorithm, after k iterations, the k vectors that span $\mathcal{K}_k$ are built and stored. If n is large and k becomes large, storage may become an issue. Often times we may only be able to store a small number of vectors in our computer's memory. In this case, we must seek alternative methods.

These methods include GMRES(m) (or GMRES with restarts), TFQMR or BiCGStab. We only mention these methods here and refer the reader to [15] or another suitable resource.

A.2 Nonlinear Iterative Methods

In this section, we'll focus on solution methods for two types of nonlinear problems, fixed-point problems and root-finding problems. The fixed-point problem will be considered first and takes the form

$$x = K(x) \tag{A.21}$$

where K is some nonlinear function. The root-finding problem will be expressed as

$$F(x) = 0. \tag{A.22}$$

It should be apparent that some nonlinear problems may lend themselves to one form or the other. At times, it may be convenient to switch forms of a problem in order to gain some algorithmic advantage.

A.2.1 Nonlinear Equation Basics

Before discussing specific methods for solving Equations A.21 and A.22, we must present a few definitions and theorems that will aid in our analysis of these algorithms. To begin, it will be important to catalogue these methods based upon their convergence rates. The following definition allows us to classify these convergence rates [15].

Definition 8. *Let $\{x_n\}$ be a sequence in $\mathbb{R}^n$ and take $x^* \in \mathbb{R}^n$. Then*

- *$x_n \to x^*$ q-quadratically if $x_n \to x^*$ and there is a constant $C > 0$ such that*

$$\|x_{n+1} - x^*\| \leq C\|x_n - x^*\|^2. \tag{A.23}$$

- *$x_n \to x^*$ q-superlinearly with q-order $\alpha > 1$ if $x_n \to x^*$ and there is a constant $C > 0$ such that*

$$\|x_{n+1} - x^*\| \leq C\|x_n - x^*\|^\alpha. \tag{A.24}$$

- *$x_n \to x^*$ q-superlinearly if*

$$\lim_{n\to\infty} \frac{\|x_{n+1} - x^*\|}{\|x_n - x^*\|} = 0. \tag{A.25}$$

- *$x_n \to x^*$ q-linearly with q-order $\beta \in (0, 1)$ if*

$$\|x_{n+1} - x^*\| \leq \beta\|x_n - x^*\|. \tag{A.26}$$

for n sufficiently large.

When considering Newton's method we must be familiar with the Jacobian matrix and the Fundamental Theorem of Calculus.

Definition 9. *Let $F(x)$ be a function from $\mathbb{R}^n$ to $\mathbb{R}^n$. If each component of F is differentiable at $x \in \mathbb{R}^n$, then we define the Jacobian matrix $J(x) = F'(x)$ by*

$$F'(x)_{ij} = \frac{\partial f_i}{\partial x_j}(x). \tag{A.27}$$

Theorem 9. *Let f be a real-valued function on the closed-interval $[a, b]$ and suppose F is an antiderivative of f in $[a, b]$. Then,*

$$F(b) - F(a) = \int_a^b f(x)\ dx. \tag{A.28}$$

However, this formulation (Theorem 9) of the Fundamental Theorem of Calculus does not suit our needs. Instead, we'll express it as follows [15]:

Theorem 10. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be differentiable in an open set $\Omega \subset \mathbb{R}^n$ and suppose $x^* \in \Omega$. Then, for all $x \in \Omega$ sufficiently close to x^*, we have*

$$F(x) - F(x^*) = \int_0^1 F'(x^* + t(x - x^*))(x - x^*)\ dt. \tag{A.29}$$

Before continuing, we must introduce the concept of Lipschitz continuity and the related concept of contraction mappings [15].

Definition 10. *Let $\Omega \subset \mathbb{R}^n$ and let $G : \Omega \to \mathbb{R}^m$. G is said to be Lipschitz continuous on Ω with Lipschitz constant γ if*

$$\|G(x) - G(y)\| \leq \gamma \|x - y\| \tag{A.30}$$

for all x, $y \in \Omega$.

Definition 11. *Let $\Omega \subset \mathbb{R}^n$. $K : \Omega \to \mathbb{R}^n$ is a contraction mapping on Ω if K is Lipschitz continuous on Ω with Lipschitz constant $\gamma < 1$.*

A.2.2 Nonlinear Fixed-Point Problems

In this section we'll discuss two methods for solving non-linear problems described by Equation A.21. The first method is aptly named *fixed-point iteration*, but has also been called *Picard iteration* throughout this thesis. We'll let x^* denote a solution to the fixed-point problem $x = K(x)$ and refer to x^* as a *fixed-point*.

Picard iteration, or fixed-point iteration, begins with some initial iterate x_0 and simply computes the successive iterates by evaluating the fixed-point map at the previous iterate,

$$x_{n+1} = K(x_n). \tag{A.31}$$

This method for solving Equation A.21 is very simple to implement, but may suffer from slow convergence. The following theorem [15] characterizes the convergence of fixed-point iteration.

Theorem 11. *Let Ω be a closed subset of $\mathbb{R}^n$ and let K be a contraction mapping on Ω with Lipschitz constant $\gamma < 1$ such that $K(x) \in \Omega$ for all $x \in \Omega$. Then there is a unique fixed-point of K, $x^* \in \Omega$, and the iteration defined by A.31 converges q-linearly to x^* with q-factor γ for all initial iterates $x_0 \in \Omega$.*

Often times, this q-linear convergence may be too slow for our given application. In this case, it would be ideal to accelerate the solution to Problem A.21. We may employ Anderson Acceleration [29] to accelerate the fixed-point iteration. Anderson Acceleration was described in Section 2.4.

Unfortunately, for general nonlinear problems, the theory for Anderson Acceleration is weak. It has been noted [24, 9] that there is a relationship between Anderson acceleration and quasi-Newton updating, however that is beyond the scope of this thesis. For linear problems it has been shown that Anderson acceleration is related to GMRES [29].

A.2.3 Root Finding

In this section we wish to solve the problem described by Equation A.22. We will use Newton's method to solve these problems. Newton's method is given by the formula

$$x_{n+1} = x_n - F'(x_n)^{-1}F(x_n) = x_n + s. \tag{A.32}$$

where s is the Newton step, given by $s = -F'(x_n)^{-1}F(x_n)$.

The main convergence results for Newton's method follows [15]:

Theorem 12. *Suppose there exists an $x^* \in \mathbb{R}^n$ such that $F(x^*) = 0$, $F' : \Omega \to \mathbb{R}^{n\times n}$ is Lipschitz continuous with Lipschitz constant γ, and $F'(x^*)$ is nonsingular. Then, there exists $K > 0$ and $\delta > 0$ such that if $\|x_n - x^*\| < \delta$ the Newton iterate from Formula A.32 satisfies*

$$\|e_{n+1}\| \leq K\|e_n\|^2.$$

While Theorem 12 does not guarantee convergence, the following theorem demonstrates local convergence.

Theorem 13. *Suppose there exists an $x^* \in \mathbb{R}^n$ such that $F(x^*) = 0$, $F' : \Omega \to \mathbb{R}^{n\times n}$ is Lipschitz continuous with Lipschitz constant γ, and $F'(x^*)$ is nonsingular. Then there is a $\delta > 0$ such that if $\|x_0 - x^*\| < \delta$, the Newton iteration*

$$x_{n+1} = x_n - F'(x_n)^{-1}F(x_n)$$

converges q-quadratically to x^*.

Generally, we terminate Newton's method when the norm of the residual is below some tolerance. However, in many instances we set a relative tolerance as well, and terminate when

$$\|F(x)\| \leq \tau_r \|F(x_0)\| + \tau_a. \tag{A.33}$$

Often times, including the entirety of this thesis, forming and storing the Jacobian matrix would require too much computation to be feasible. When computing the Newton step, s, we never actually invert $F'(x_n)^{-1}$. Instead, we solve the following linear system for s,

$$F'(x_n)s = -F(x_n). \tag{A.34}$$

Fortunately, with this formulation of the problem, we never need to compute $F'(x_n)$. Instead, we compute the Jacobian-vector product, $F'(x_n)s$. This product can be approximated by

$$F'(x_n)s = \frac{F(x_n + hs) - F(x_n)}{h}. \tag{A.35}$$

When the only error in the computation of F is machine roundoff, ε_{mach}, we generally choose $h = \sqrt{\varepsilon_{mach}}$ or $h = 10^{-7}$ [15].

Now, Equation A.34 can be solved using our choice of Krylov methods. Recall from our discussion on Krylov methods that they do not require a representation of the matrix, only a matrix-vector product routine. Since our solution to Equation A.34 is no longer "exact," we expect that we may take a small performance hit in the convergence of Newton's method. This is, in fact, the case [15] and is characterized by Theorem 14.

Theorem 14. *Suppose there exists an $x^* \in \mathbb{R}^n$ such that $F(x^*) = 0$, $F' : \Omega \to \mathbb{R}^{n \times n}$ is Lipschitz continuous with Lipschitz constant γ, and $F'(x^*)$ is nonsingular. Then there is a $\delta > 0$ such that if $\|x_0 - x^*\| < \delta$, $\{\eta_n\} \subset [0, \eta]$ with $\eta < \bar{\eta} < 1$, then the inexact Newton iteration*

$$x_{n+1} = x_n + s$$

where

$$\|F'(x_n)s_n + F(x_n)\| \leq \eta_n \|F(x_n)\|$$

converges q-linearly with respect to $\|\cdot\|$ to x^*. Furthermore,

- *if $\eta_n \to 0$, then the convergence is q-superlinear, and*
- *if $\eta_n \leq K_\eta \|F(x_n)\|^p$ for some $K_\eta > 0$, then the convergence is q-superlinear with q-order $1 + p$.*

In some cases, it may be difficult to compute or find an x_0 such that $\|x_0 - x^*\| < \delta$ so that the iteration converges as the theorems predict. In this case, we add a line-search to the algorithm. In general, the Newton step may point in the correct general direction of the root, however the magnitude of the Newton step may be far too large. The line-search allows us to take a smaller step in the same direction. The algorithm for the line search is given in Algorithm A.2.3.

Algorithm A.2.3

Find s such that $\|F'(x_n)s + F(x_n)\| \leq \eta\|F(x)\|$
Set $\lambda = 1$
while $\|F(x_n + s)\| > (1 - \alpha\lambda)\|F(x)\|$ AND $\lambda \geq \lambda_{min}$ **do**
  $\lambda = \lambda/2$
end while
if $\lambda < \lambda_{min}$ **then**
  Terminate with Line Search Failure
else
  Set $x_{n+1} = x_n + \lambda s$.
end if

This is a very simple version of the line search. In [15], the α parameter is chosen to be 10^{-4}. One could choose to cut down the step length by any fraction σ. In this case, we'll replace the $\lambda = \lambda/2$ step length reduction by $\lambda = \sigma\lambda$ for $\sigma \in (0, 1)$.

The line search should only activate during the initial phase of the iteration. Once our iterates have entered the region of local convergence, the line search should never reduce step lengths and we should see quadratic convergence for the remainder of the iteration.

Appendix B

Numerical Eigenvalue Computations

Suppose we wish to solve the eigenvalue problem,

$$Ax = \lambda x, \tag{B.1}$$

in which A has spectrum $\sigma = \{\lambda_i\}_{i=1}^n$ and $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$. Throughout this section we will assume that A is a nonsingular matrix so that $|\lambda_n| > 0$. Furthermore, suppose that the dimension of the eigenspace associated with λ_1 is 1. In this case, we can use the *Power Method* [26] to approximate λ_1, and its associated eigenvector v_1.

Algorithm B: Power Method

Choose approximate eigenvector $\vec{x_0}$.
$n = 0$
while eigenvalue not converged **do**
  Set $y_{n+1} = Ax_n$.
  Set $\mu_{n+1} = \alpha$, where α is the element of y_{n+1} with maximum magnitude.
  Set $x_{n+1} = \frac{1}{\alpha} y_{n+1}$.
  Increment $n = n + 1$
end while

Theorem 15. *Let $\{v_m, \lambda_m\}_{m=1}^n$ be the set of eigenpairs of A and suppose λ_1 is the eigenvalue of A with maximum magnitude and that nullity$(A - \lambda_1 I) = 1$. Furthermore, suppose*

$$x_0 = \sum_{i=1}^{n} \beta_i v_i \tag{B.2}$$

in which $\beta_1 \neq 0$. Then Algorithm B builds a sequence of vectors x_n and scalars μ_n such that

$x_n \to v_1$ *and* $\mu_n \to \lambda_1$ *as* $n \to \infty$.

Proof. Let us begin by considering repeated applications of the matrix A to the vector x_0,

$$\begin{aligned} A^k x_0 &= A^k \sum_{i=1}^{n} \beta_i v_i \\ &= \sum_{i=1}^{n} \beta_i A^k v_i \\ &= \sum_{i=1}^{n} \beta_i \lambda_i^k v_i \\ &= \lambda_1^k \sum_{i=1}^{n} \beta_i \left(\frac{\lambda_i}{\lambda_1}\right)^k v_i \end{aligned} \tag{B.3}$$

As k tends to infinity, we have

$$\left(\frac{\lambda_i}{\lambda_1}\right)^k \to 0$$

for all $i \neq 1$. Therefore, we have

$$\lim_{k\to\infty} A^k x_0 = \lim_{k\to\infty} \beta_1 \lambda_1^k v_1. \tag{B.4}$$

Clearly, this sequence diverges if $\lambda_1 \geq 1$.

However, in Algorithm B, the normalization constant μ_k ensures that we have

$$\|x_k\|_\infty = 1$$

for all k. Thus, the sequence of iterates

$$x_1 = \frac{Ax_0}{\|Ax_0\|}, x_2 = \frac{Ax_1}{\|Ax_1\|}, \ldots, x_n = \frac{Ax_{n-1}}{\|Ax_{n-1}\|}, \ldots \tag{B.5}$$

converges to v_1 in which $\|v_1\| = 1$. We can see that $\mu_n \to \lambda_1$ by noticing

$$\begin{aligned} \lambda_1 v_1 &= Av_1 \\ \lambda_1 \|v_1\| &= \|Av_1\| \\ \lambda_1 &= \|Av_1\|. \end{aligned}$$

□

Equation B.3 allows us to determine the rate of convergence for the power method quite

easily by recalling that $\left(\frac{\lambda_i}{\lambda_1}\right)^k$ tends to zero for all $i \neq 1$. Therefore, we can conclude that this method converges at a rate of $\mathcal{O}(|r|^k)$, in which

$$r = \frac{\lambda_2}{\lambda_1} \tag{B.6}$$

where we have ordered the eigenvalues in order of decreasing magnitude.

We could also apply the power method to A^{-1}. The eigenpairs of A^{-1} are $\{v_i, \frac{1}{\lambda_i}\}_{i=1}^n$. Here, the v_i's and λ_i's are the same eigenvectors and eigenvalues as described for the matrix A. In this case, the power method applied to A^{-1}, known as the *Inverse Power Method*, will converge to the eigenvalue with the smallest magnitude, λ_n, and its associated eigenvector v_n at a rate

$$\mathcal{O}\left(\left|\frac{\frac{1}{\lambda_{n-1}}}{\frac{1}{\lambda_n}}\right|^k\right) = \mathcal{O}\left(\left|\frac{\lambda_n}{\lambda_{n-1}}\right|^k\right). \tag{B.7}$$

Furthermore, suppose we wish to compute an arbitrary eigenvalue, λ_j, and we have some approximation to this eigenvalue, σ, which satisfies

$$|\lambda_j| = \min |\lambda_i|. \tag{B.8}$$

It is easy to see that the set of eigenpairs for $A - \sigma I$ is $\{v_i, \lambda_i - \sigma\}_{i=1}^n$. In this case, the inverse power method applied to $A - \sigma I$ will converge to $\eta_j = \lambda_j - \sigma$. We can see that this method will converge at a rate

$$\mathcal{O}\left(\left|\frac{\lambda_j - \sigma}{\lambda_c - \sigma}\right|^k\right), \tag{B.9}$$

where

$$|\lambda_c - \sigma| = \min_{i \neq j} |\lambda_i - \sigma|. \tag{B.10}$$

If σ is chosen very close to λ_j we can expect especially fast convergence. This method is known as the *Shifted Inverse Power Method*.

This allows us to accelerate convergence to the dominant eigenvalue, λ_1, if we can come up with a reasonably accurate approximation. For example, consider applying the power method to a matrix A in which $\lambda_1 = .995$ and $\lambda_2 = .990$. The dominance ratio for this matrix is

$\rho \approx .99497$. Suppose we wish to decrease the error by a factor of 10^{-5}. Then, we have

$$\begin{aligned}
\|e_k\| &\leq \left(\frac{\lambda_2}{\lambda_1}\right)^k \|e_0\| \\
\frac{\|e_k\|}{\|e_0\|} &\leq \left(\frac{\lambda_2}{\lambda_1}\right)^k \\
\log\left(\frac{\|e_k\|}{\|e_0\|}\right) &\leq k \log\left(\frac{\lambda_2}{\lambda_1}\right) \\
\frac{\log\left(\frac{\|e_k\|}{\|e_0\|}\right)}{\log\left(\frac{\lambda_2}{\lambda_1}\right)} &\leq k.
\end{aligned}$$

Substituting λ_1, λ_2, and our desired relative reduction in error suggests we will need 2286 iterations to converge.

Let us consider the same example, but apply the shifted inverse power method using $\sigma = 1$. Then, we have $\lambda_1 - \sigma = -.005$ and $\lambda_2 - \sigma = -.010$. This effectively has changed the dominance ratio to $\rho = .5$. Now, we can achieve the desired error reduction in only 17 iterations. While this example was built specifically to display the potential advantages of the shifted inverse power method, we often see a drastic reduction in the number of iterations in practice as well.

Appendix C

Monte Carlo Error vs. Truncation Error

When developing hybrid methods, we noted that the motivating reason for turning to stochastic methods is the lack of discretization error. Monte Carlo methods can be free of spatial, angular and energy group discretization error. By attempting to accelerate these Monte Carlo solutions using deterministic acceleration techniques, an interesting question arises. Has the low-order system introduced a spatial discretization error that pollutes the pure Monte Carlo calculation?

We'll attempt to quantify both the discretization error which arises from the solution to the low-order equation and the Monte Carlo error. If the discretization error from the low-order problem is considerably less than the Monte Carlo error, we believe that this discretization error will be lost in the noise of a Monte Carlo simulation.

C.1 Error Quantification Methods

In this section, we will discuss the numerical methods which we use in an attempt to compute the constants in the error terms for both the low-order discretization error and the Monte Carlo noise. It is important to note that we are not trying to derive exact error estimates. For this analysis, it is more important to gain approximate error bounds or obtain the order of magnitude of the error.

C.1.1 Low-Order Truncation Error

We know that the discretization used for the low-order problem is second-order accurate. Therefore, for a given problem, the error behaves like

$$E(h) = C \cdot h^2.$$

We'd like to be able to approximate the value of C.

In order to do this, we will work under the assumption that the value of $\hat{D}$ is known everywhere throughout the domain. We can realize this assumption to high-accuracy by computing a very high-accuracy solution to the transport equation using an extremely fine mesh, with mesh width h. On this fine mesh, we will also compute the value of $\hat{D}$. We will use the solution ϕ^* on this mesh as a "true solution" for subsequent calculations.

Now, we will choose a series of mesh widths, $\cdots > 16H > 8H > 4H > 2H > H \gg h$. On each of these meshes, we will solve the low-order problem and compare their solutions to the benchmark solution, ϕ^*. To solve each of these low-order problems, we will evaluate $\hat{D}$ from the fine-mesh solution at the appropriate grid points. We'll use the true solution and current to provide boundary conditions as well.

C.1.2 Monte Carlo Error

In 1-D, the Monte Carlo error is determined mostly by the length of each cell in terms of mean free paths and the number of particles used. Furthermore, the Monte Carlo error is directly proportional to the strength of the source term.

In order to quantify the Monte Carlo error, we will compute the error which results from solving

$$\begin{aligned} \mu\frac{\partial\psi}{\partial x} + \Sigma_t\psi &= Q(x) \\ Q(x) &= 1 \end{aligned}$$

for various values of the spatial mesh width (h), length of the medium (τ), number of particles (N_{MC}) and total cross-sections (Σ_t). We expect the error to decay like the variance,

$$E(N_{MC}, h, \tau, \Sigma_t) = C(h, \tau, \Sigma_t) \cdot \frac{1}{\sqrt{N_{MC}}}.$$

For each combination of h, τ and Σ_t that we investigate, we will run simulations using 10^5, 10^6, 10^7, 10^8, 10^9 and 10^{10} particles per transport sweep. We will run 20 simulations with each particle count and use a weighted average of all of the simulations to compute a true solution, ϕ^*. Then, for each of the simulations, we will use this true solution to compute an error, $\|\phi^* - \phi\|_\infty$. We will then compute an average error associated with each particle count. Using these average errors, we will attempt to approximately back-out the constant in the Monte Carlo error term.

C.2 Numerical Results

First, we will display the computational results for the low-order truncation error. The problems which we tested are given by

$$\mu \frac{\partial \psi}{\partial x} + \psi = \frac{1}{2} \left[c\phi + Q(x) \right]$$

where $Q \equiv 1$. We let the scattering ratio, c, and the length of the domain vary. The error term constants for various problems have been computed and are displayed in Table C.1.

Table C.1: The LO truncation error constant is determined by the scattering ratio and the length of the domain in terms of mean free paths.

Scattering Ratio	Length of Domain (MFPs)	$\|\phi^*\|_\infty$	Error Term Constant
.7	.001	.0035	.0324
.7	.01	.029	.12
.7	.1	.19	.145
.7	1	1.21	.14
.7	10	3.29	.09
.9	.001	.0035	.108
.9	.01	.030	1.4
.9	.1	.20	.04
.9	1	1.55	.075
.9	10	9.04	.072
.99	.001	.0035	.00109
.99	.01	.030	.004
.99	.1	.21	.0053
.99	1	1.78	.0095
.99	10	34.5	.0044

We plot this data and data for additional scattering ratios in Figure C.1. We can see that the constant is very tightly correlated to both the scattering ratio and the length of the domain. We notice that for domain lengths less than a mean free path, if the absorption ratio $(1 - c)$ increases by a factor of 10, the error term constant increases by roughly a factor of 10. Beyond one mean free path, the error constants all appear to trend towards a value between .001 and .01. With a constant source term, as the domain gets larger the solution approaches a constant value in the interior of the domain, sloping off sharply near the boundaries. For these problems, the largest error occurs in these regions where the scalar flux changes rapidly.

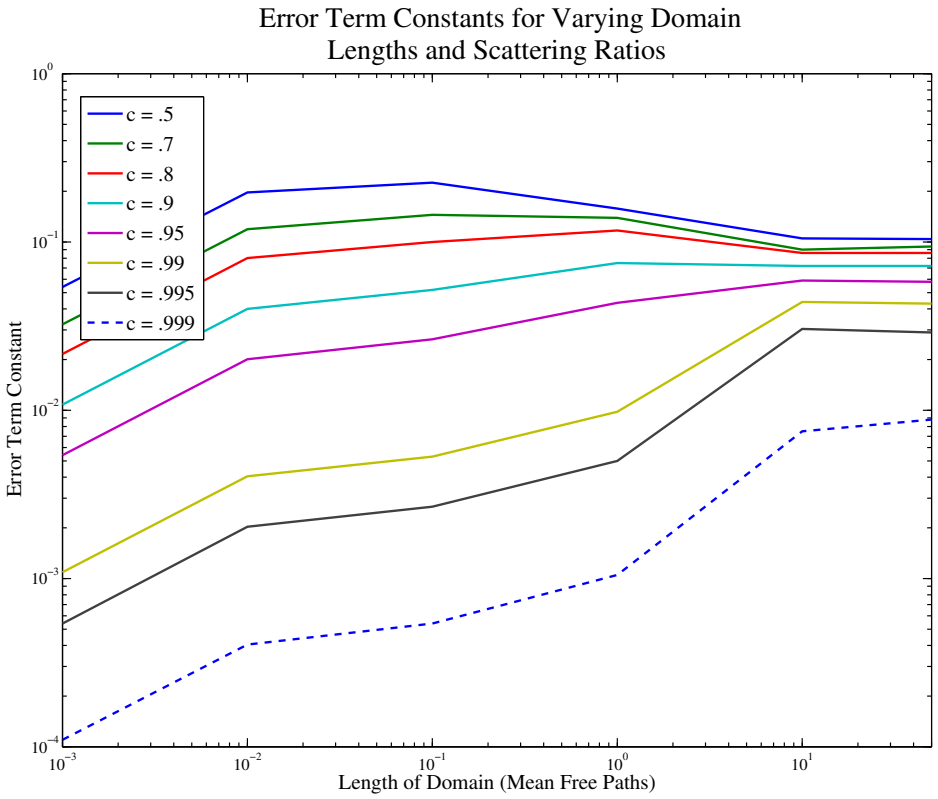

Figure C.1: Low-Order Solution Error Term Constants as a function of the scattering ratio and domain length.

Now, we turn our attention to the computation of the constant in the Monte Carlo error term. In the computation of these error constants we have held $\Sigma_t = 1$. Changing Σ_t only changes the length of the domain in terms of mean free paths, which is already being controlled by τ. In the case of a heterogenous domain, both Σ_t and τ should be varied since cells in different areas of the domain will have different lengths in terms of mean free paths. In Table C.2, we display the computed constant in the error term:

Table C.2: Monte Carlo error term constants are determined by the length of the medium and the number of cells used.

$N_x \backslash \tau$	1	10	100	1000	10000
10	2.1	4.2	5.2	4.8	4.7
100	4.5	11	20	22	23
1000	7.2	19	46	79	90
10000	11.1	28	80	170	298

This data in this table behaves like expected. By increasing τ each cell gets wider and each particle can contribute less information to the tally in that cell. Furthermore, as N_x increases, the cell average flux has less particles contributing to the tallies. Thus, as N_x and τ increase, so does the constant in the error term. These error term constants have been plotted on a log-log-log scale in Figure C.2.

Tables C.1 and C.2 allow us to get a rough estimate of the number of particles per Monte Carlo transport sweep we need to use in order to match the Monte Carlo error and the LO truncation error for a given configuration (scattering ratio, length of domain, and number of cells).

For example, suppose we wish to solve the transport equation for $c = .9$ and $\tau = 10$ using 100 cells. In this case, the LO truncation error term constant is .072. So, using 100 cells we have $h = .1$, and the LO truncation error is

$$E_{LO} \approx .072 \times h^2 = .00072.$$

Now, the Monte Carlo error term for $\tau = 10$ and $N_x = 100$ is roughly 11 for a unit source. Since we know that near the solution, the scalar flux grows as large as 9.04, our source term can be bounded by

$$S = \frac{1}{2}[c\phi + Q] < \frac{1}{2}[.9 \cdot .904 + 1] < 4.57.$$

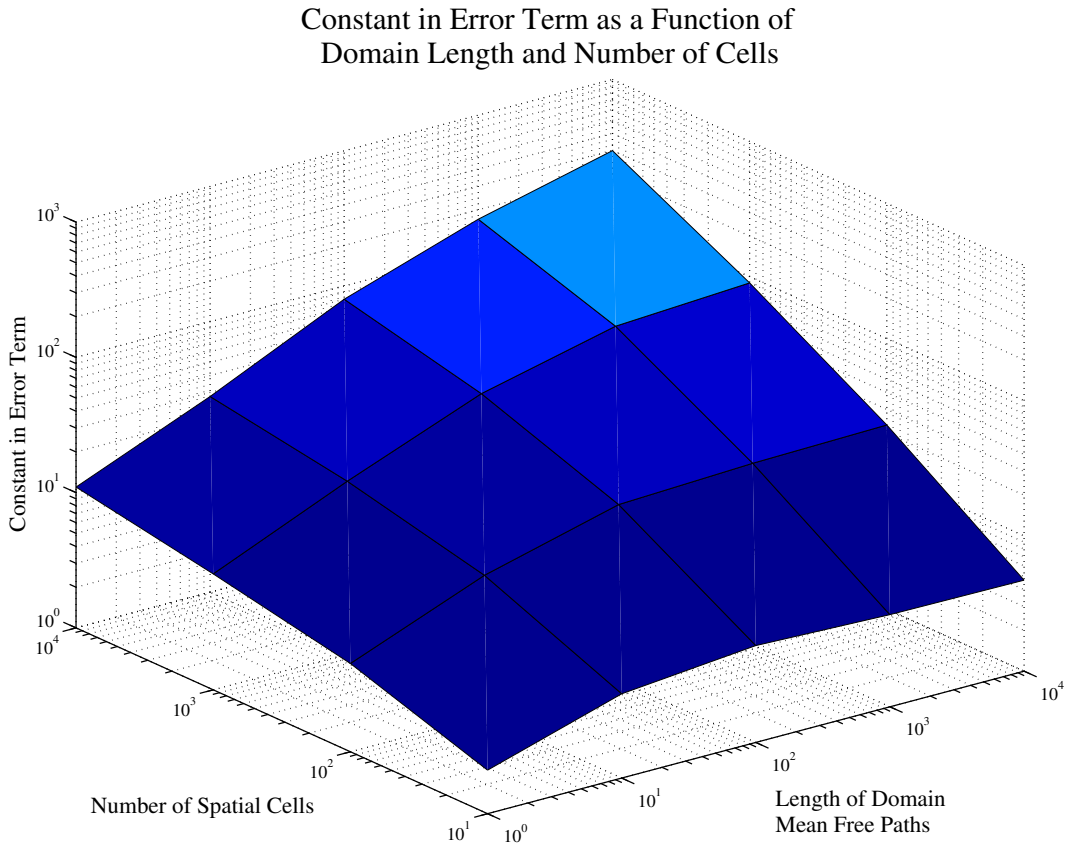

Figure C.2: Monte Carlo Error Term Constants as a function of the number of cells and domain length.

Now, we seek a number of particles N such that

$$.0072 \approx \frac{11 \times 4.57}{\sqrt{N}}.$$

Solving for N tells us we'd need roughly 4.87×10^9 particles to achieve a similar level of error in both the LO solve and the Monte Carlo transport sweep. It is important to note that the bound on the scalar flux need only be approximate, so using a diffusion solution in order to evaluate this value would be practical, since the true solution is unknown.

If we decide that the error in this simulation would be too large, we increase the number of spatial cells and re-compute the number of particles required to achieve a Monte Carlo error at the same level of the truncation error. Suppose we do this for the same problem with $N_x = 1000$. In this case, it would require 1.45×10^{14} particles to match the errors. This number of particles is probably impractical, so for this configuration, we would be guaranteed that the LO truncation error is far smaller than the Monte Carlo error near the solution.

Appendix D

Pure Monte Carlo Computations

In this Appendix, we'll consider the computation of the k-eigenvalue for one of the 1-D Test problems consider in Chapters 4 and 6. The cross-sections and domain parameters are given by:

$$\begin{aligned}
&\Sigma_t = 1 \quad \Sigma_s = .856 \quad \nu\Sigma_f = .144 \qquad \text{(fissile region)} \\
&\Sigma_t = 1 \quad \Sigma_s = .856 \quad \nu\Sigma_f = 0 \qquad \text{(reflector region)} \\
&\tau_f = 50 \ \text{ (length of fissionable region)} \\
&\tau_r = 5 \ \text{ (length of reflector region)}
\end{aligned}$$

We estimate the dominance ratio for this problem to be roughly $\rho = .94$. As far as eigenvalue problems go, this dominance ratio is on the low side, as many times we see dominance ratios that exceed .99.

Even at this low dominance ratio, analog Monte Carlo requires an abundance of particle histories to compute the eigenvalue. This is demonstrated in Figures D.1 - D.8. In Figures D.1, D.4 and D.7, we plot the history of the eigenvalue, the computed eigenvalue and two lines representing a relative error of 10^{-5} above and below the computed eigenvalue. In Figures D.2, D.5 and D.8, we plot the history of the Shannon Entropy over the active cycles.

For these tests, we used 10^5, 10^6 and 10^7 particles per active cycle, respectively. At the end of each of these three simulations, a total of 4×10^9, 10^{10} and 10^{10} particles per simulation, respectively. On average, each particle undergoes 6.9 flights before leaving the medium or being absorbed. As we can see, for each of these tests, the Shannon entropy is still slowly changing, as is the eigenvalue. In Figures D.3, D.6, and D.9 we see that the change in the Shannon Entropy from cycle to cycle is decreasing as the iteration progresses. Using 10^7 particles per active cycle, the eigenvalue appears to be nearly converged.

Given these results, we can make a few comments regarding pure Monte Carlo eigenvalue

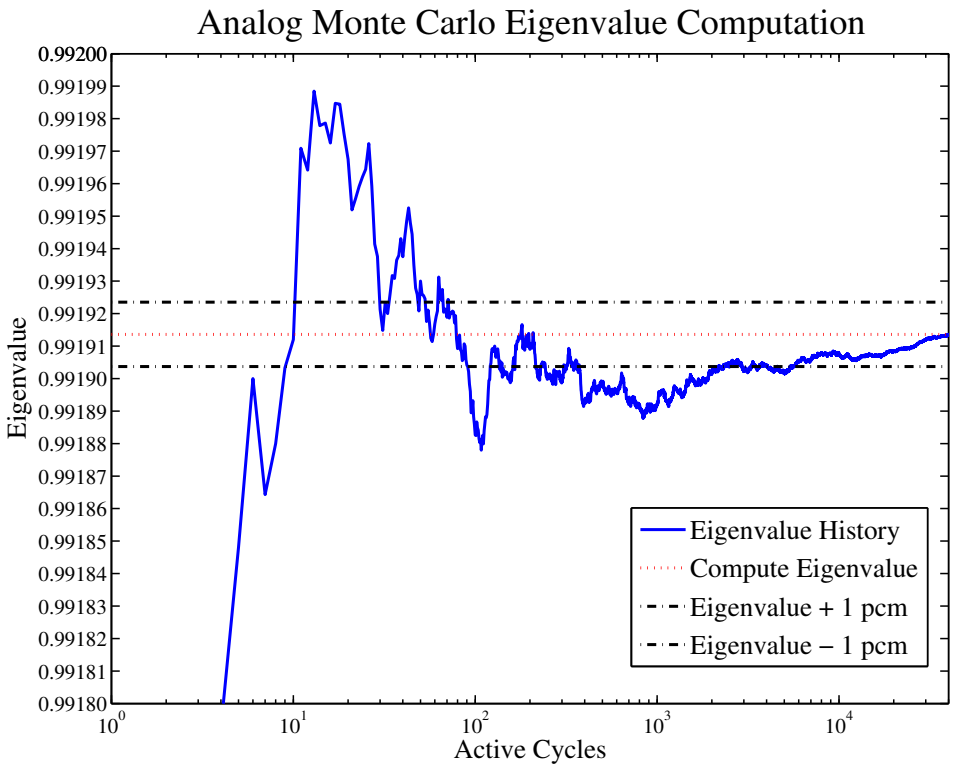

Figure D.1: Convergence history of the eigenvalue using 40000 active cycles with 10^5 particles per cycle.

computations. First of all, it can be difficult to determine when an analog Monte Carlo computation has converged. Here we have no nonlinear residual which measures how close the computed eigenpair is to the true solution. Furthermore, often times at early stages of the iteration, two consecutive computations of the eigenvalue may fall extremely close to one another, however the eigenvalue is far from converged.

Secondly, the Shannon entropy is a good metric for convergence. As the Shannon entropy stops changing from cycle to cycle, so does the eigenvalue. Lastly, it is hard to draw head to head comparisons between hybrid methods and pure Monte Carlo computations. At the end of these three simulations, a total of roughly 2.8×10^{10}, 6.9×10^{10}, and 6.9×10^{10} particle flights have been simulated. When solving the problem using hybrid methods, a total of 3.2×10^{10}, flights had been simulated. However, using the hybrid method, we have more confidence that the eigenvalue has converged to the desired tolerance. These eigenvalues agree to four digits which gives us confidence that the low-order mesh has not injected significant discretization error.

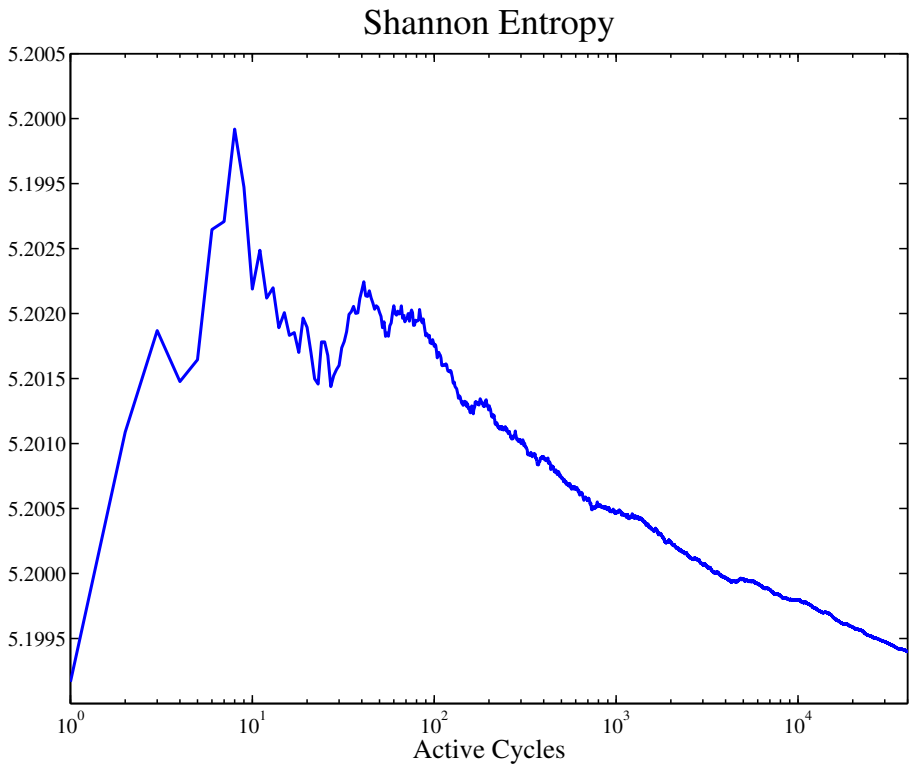

Figure D.2: Shannon Entropy history using 40000 active cycles with 10^5 particles per cycle.

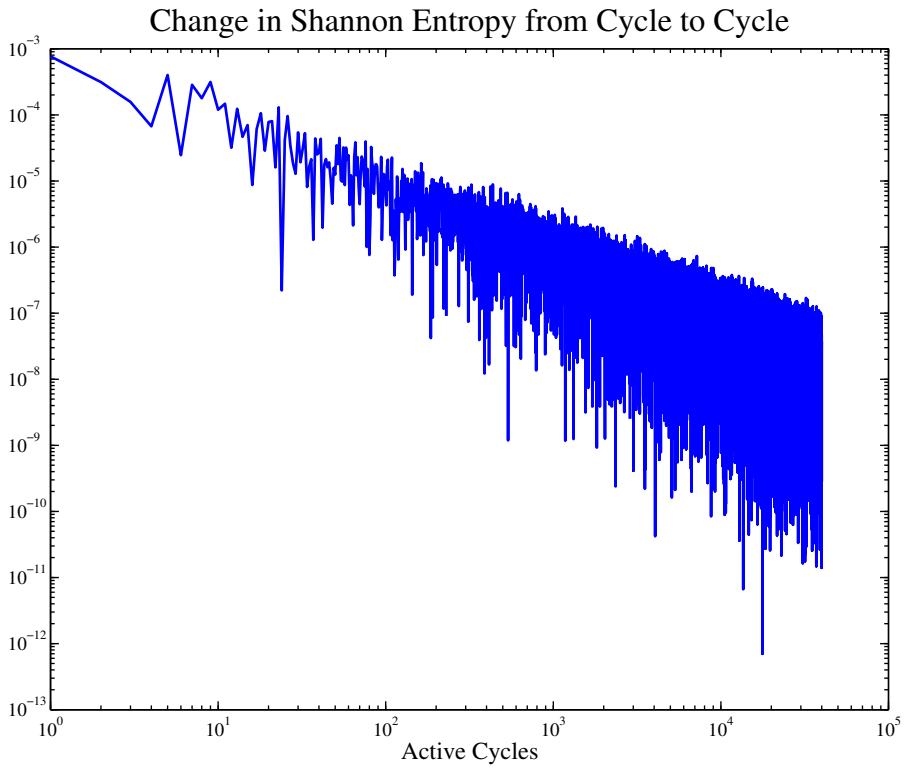

Figure D.3: Change in Shannon Entropy from cycle to cycle using 40000 active cycles with 10^5 particles per cycle.

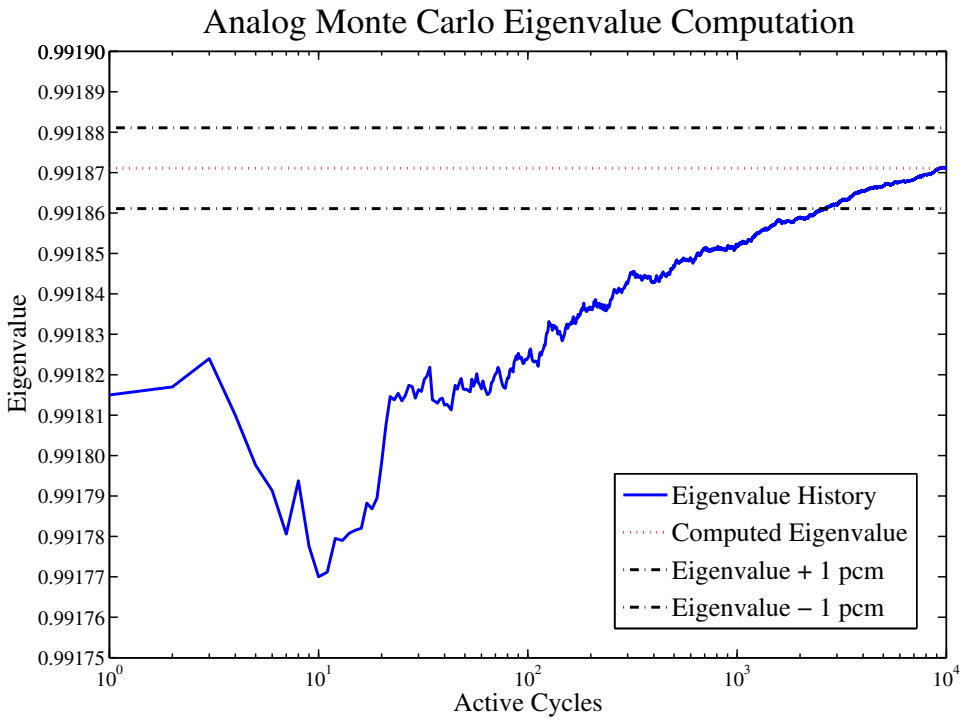

Figure D.4: Convergence history of the eigenvalue using 10000 active cycles with 10^6 particles per cycle.

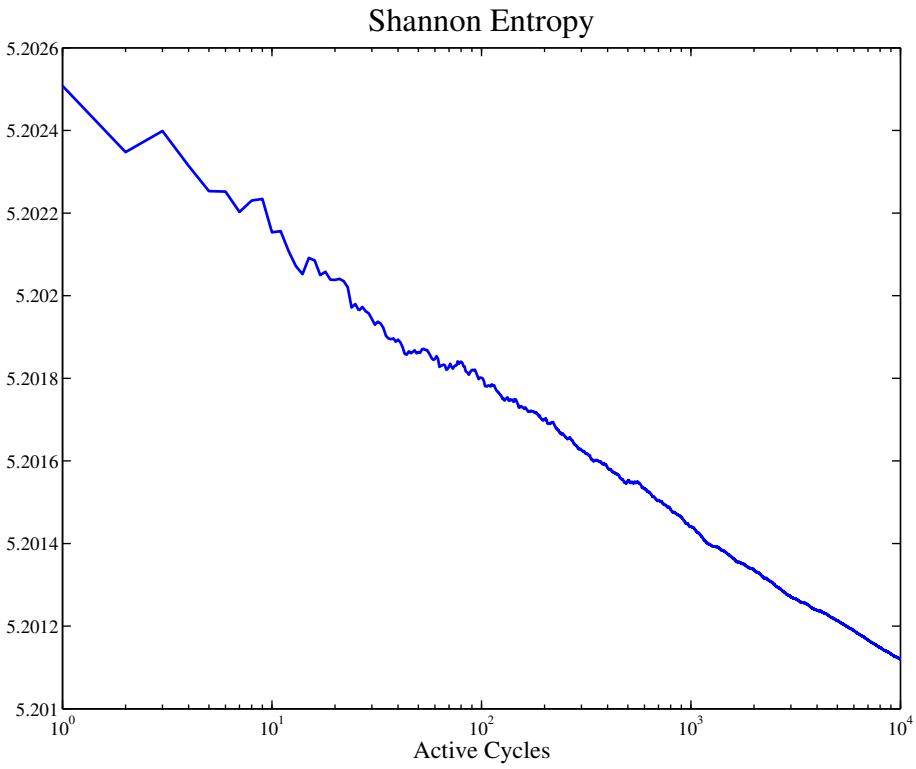

Figure D.5: Shannon Entropy history using 10000 active cycles with 10^6 particles per cycle.

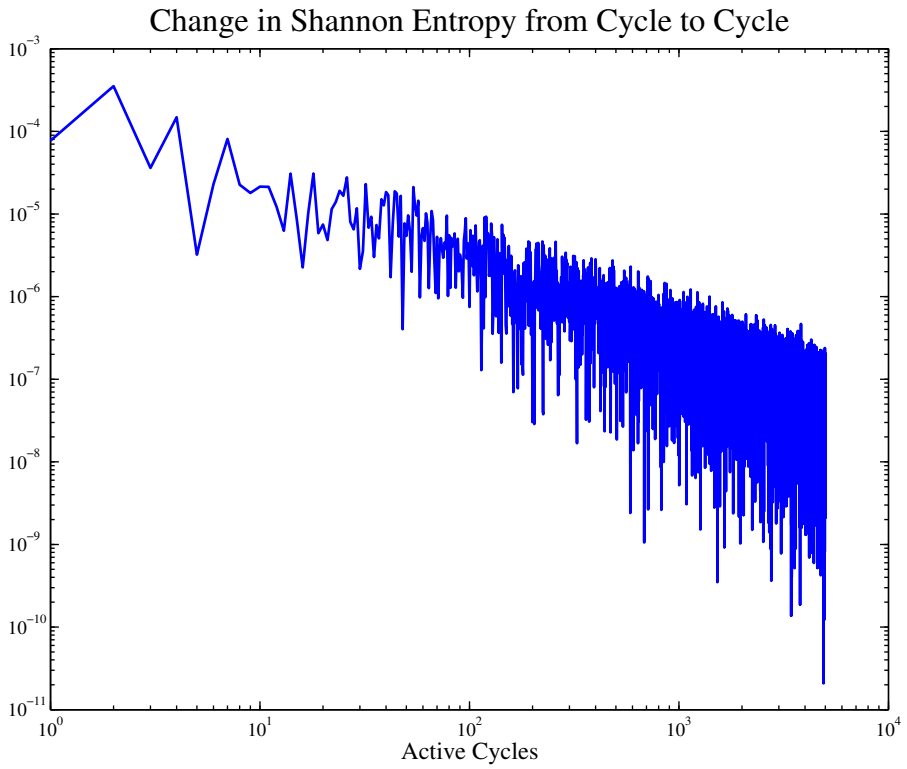

Figure D.6: Change in Shannon Entropy from cycle to cycle using 5000 active cycles with 10^6 particles per cycle.

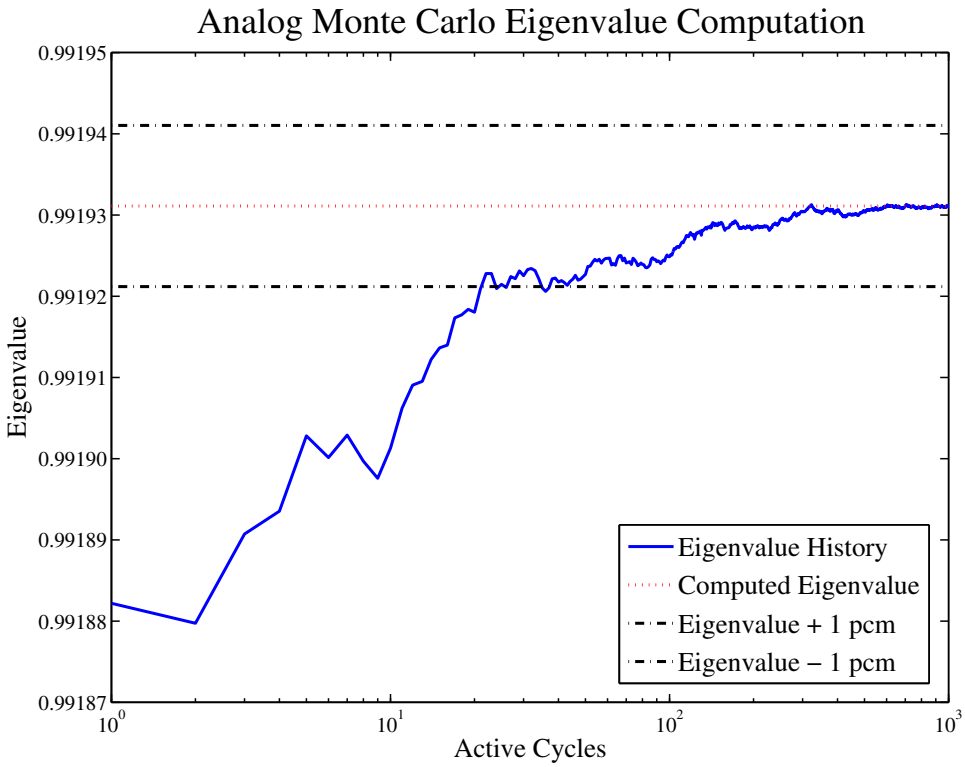

Figure D.7: Convergence history of the eigenvalue using 1000 active cycles with 10^7 particles per cycle.

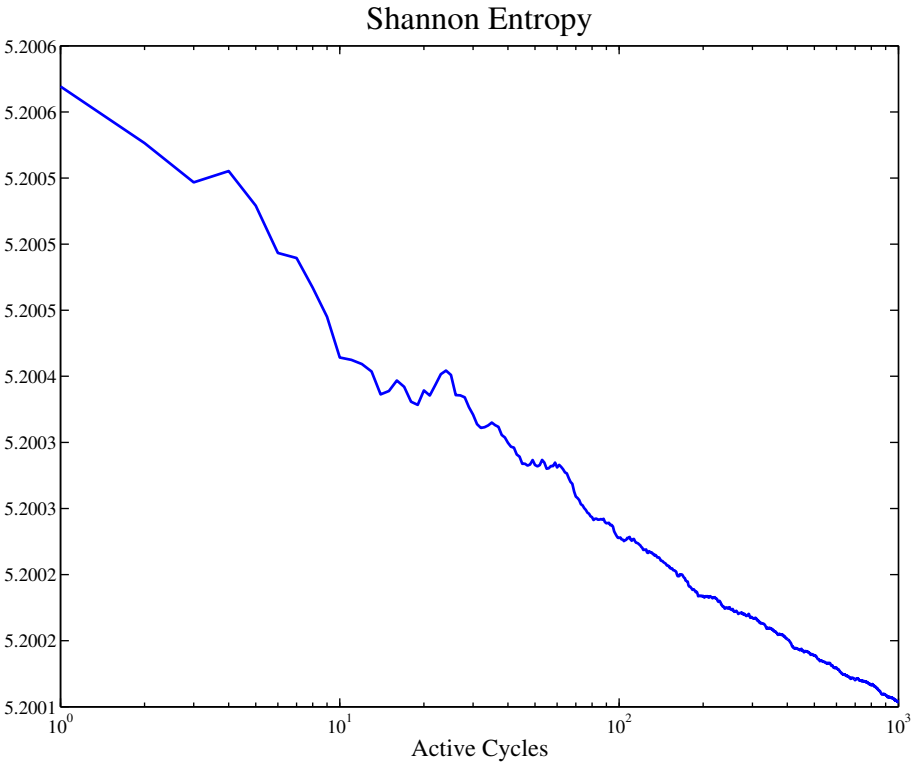

Figure D.8: Shannon Entropy history using 1000 active cycles with 10^7 particles per cycle.

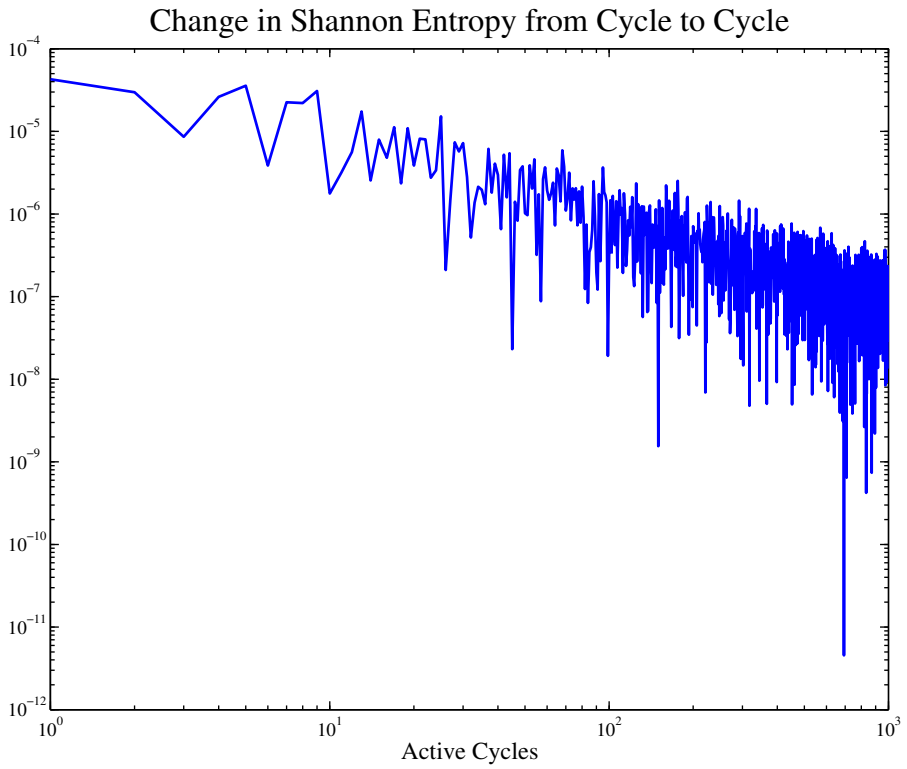

Figure D.9: Change in Shannon Entropy from cycle to cycle using 1000 active cycles with 10^7 particles per cycle.